Calcolatori Elettronici - Architettura e Organizzazione

Appendice C

Architettura x86

Giacomo Bucci

Revisione del 31 marzo 2017

Questo documento è una appendice al volume
Calcolatori Elettronici - Architettura e Organizzazione
IV edizione
McGraw-Hill Education (Italy), S.r.l.
Milano, 2017

Storia degli aggiornamenti

Marzo 2017: primo rilascio.

C

L’architettura x86

OBIETTIVI

- Presentare l’architettura $\times$86 a partire dal processore 8086
- Mostrare l’evoluzione, tecnologica e architetturale, fino ai processori correnti
- Illustrare gli aspetti caratteristici dell’architettura, con particolare riferimento alla memoria segmentata
- Esaminare l’evoluzione dell’architettura dalla versione a 16 bit a quella a 32 bit, a quella a 64 bit.
- Mostrare come è stato attuato lo “spazio lineare” della versione a 64 bit.

CONCETTI CHIAVE

Architettura $\times$86, modello di programmazione, segmentazione dello spazio di memoria, organizzazione della memoria, modalità di indirizzamento, evoluzione tecnologica, architettura a 32 bit, architettura a 64 bit..

INTRODUZIONE

La sigla $\times$86 viene usata in senso ampio, per designare la famiglia di processori che derivano dall’originale CPU 8086, tipica architettura CISC, introdotta dall’Intel nel 1978. Alternativamente si usa il termine “architettura ’86”, talvolta “architettura Intel”. Nel 1985 l’architettura è stata estesa da 16 a 32 bit; nel 2003 è stata estesa a 64 bit, questa volta non da Intel, ma da AMD. Per le corrispondeti architetture i due produttori usano rispettivamente le sigle IA-32 e Intel-64, ovvero AMD-32 e AMD-64. Noi useremo le sigle $\times$86-32 e $\times$86-64.

A partire dall’impiego nei PC portatili a finire con quello nei supercomputer, l’architettura $\times$86 non ha praticamente rivali (si vedano ad esempio i dati riportati nel testo al primo capitolo circa la diffusione nel segmento dei supercomputer). C’è una probabilità prossima al 100% che il PC di chi sta leggendo queste righe contenga un processore $\times$86. Per questo motivo si ritiene che l’architettura $\times 86$ meriti un approfondimento particolare.

La prima parte di questa appendice presenta il processore 8086 in modo relativamente approfondito, in quanto questa CPU ha condizionato per ragioni di retrocompatibilità tutti gli sviluppi successivi. Solo dopo aver mostrato l’8086 vengono illustrate le estensioni a 32 e 64 bit. Si evita fare la storia dei differenti modelli, per i quali si invita a riferirsi ai manuali dei costruttori, facilmente scaricabili da Internet. In caso di dubbio, soltanto questi sono i depositari della verità circa le rispettive macchine.

C.1 Le ragioni di un grande successo

Obiettivo dichiarato dall'Intel al momento dell'introduzione della CPU 8086 (1978) era la realizzazione di un microprocessore che migliorasse, di un ordine di grandezza, le prestazioni del precedente micro 8080/8085, mantenendo con questo una parvenza di compatibilità[1].

Fu il primo micro della seconda generazione (registri e bus dati a 16 bit, spazio degli indirizzi di almeno 1 MB) a essere disponibile sul mercato, precedendo di circa un anno i diretti concorrenti: lo Z8000 e il 68000. Ciò permise all'Intel di conquistarsi una larga fetta del mercato professionale e industriale, anche perché la società fu attenta al problema della produzione del software in ambiente industriale, accompagnando l'introduzione del micro con la disponibilità di un linguaggio di programmazione di alto livello, il PL/M-86, una sorta di PL/I[2] per microprocessori, con il quale si riducevano fortemente i tempi di sviluppo rispetto all'impiego, allora largamente diffuso, del linguaggio Assembler. Il primo manuale sull'8086 del 1979 dedicava un numero non trascurabile di pagine all'impiego del PL/M-86, cosa insolita per l'epoca[3].

Ma la famiglia 8086 si impose in modo schiacciante con l'8088, versione a 8 bit dell'8086, introdotta dall'Intel più di un anno dopo l'8086. L'8088 era una CPU completamente compatibile con l'8086, con un parallelismo interno di 16 bit, capace di indirizzare fino a 1 MB di memoria, ma il bus dati esterno è di 8 bit. L'8088 aveva le prestazioni di un micro a 16 bit, ma il fatto di avere un bus dati a 8 bit permetteva di ridurre i costi dell'elettronica. Del resto, le periferiche impiegabili su Personal Computer erano praticamente solo a 8 bit, mentre la velocità dell'8088 era perfino debordante per le tipiche applicazioni dell'epoca.

Nel 1981 l'IBM introduceva un personal computer, il PC IBM, basato sul micro 8088, con frequenza di clock pari a 4,77 MHz.

Il PC IBM non era il primo personal computer a fare la sua comparsa sul mercato. Da anni era in commercio una varietà di calcolatori personali basati su microprocessori a 8 bit del tipo 8085 o Z80. Queste macchine impiegavano normalmente il sistema operativo CP/M, lo standard dell'epoca, che sembrava non temere concorrenza e di certo nessuno ne avrebbe profetizzato una rapida scomparsa.

Su un percorso proprio procedeva la società Apple, che produceva il calcolatore personale Apple II, basato sul micro 6502 (Rockwell), pure a 8 bit, dotato di un proprio sistema operativo.

Nello stesso periodo in cui l'IBM introduceva il PC, la Digital Equipment (DEC) immetteva nel mercato un calcolatore personale (detto Rainbow) che, dentro la stessa scatola, conteneva una CPU Z80 e una CPU 8088. Con tale macchina, la Digital, allora secondo colosso informatico, preceduta solo dall'IBM come fatturato, intendeva mantenere una totale compatibilità con il mondo CP/M a 8 bit e, al tempo stesso, percorrere

[1]In pratica una vera compatibilità non esisteva, anche se essa ha avuto qualche conseguenza sulle caratteristiche architetturali della CPU 8086. La successiva evoluzione dimostrò che quello della compatibilità con il precedente mondo a 8 bit era un problema inesistente.

[2]il PL/I era un linguaggio di alto livello, all'epoca usatissimo sui sistemi IBM

[3]Il linguaggio C non aveva ancora raggiunto la popolarità attuale. I produttori di microprocessori al massimo facevano qualche accenno alla programmazione in linguaggio assembler, senza nascondere il fastidio di dover dedicare spazio a questioni considerate insignificanti rispetto a quelle relative all'architettura del microprocessore e al suo impiego nella progettazione elettronico-sistemistica.

la strada dei 16 bit e dello spazio di memoria superiore ai 64 KB[4]. Seppure tecnicamente superiore e sebbene prodotto da un costruttore di tutto rispetto, questo calcolatore personale non ebbe un successo degno di nota, schiacciato, come tutti gli altri calcolatori personali, dall'affermarsi del PC IBM.

Per farsi un'idea del perché il PC IBM abbia goduto di tanto favore, bisogna tener presente che, fino agli inizi degli anni ottanta, negli USA, che in pratica rappresentavano in modo quasi esclusivo il mercato dei calcolatori, la sigla "IBM" e la parola "calcolatori" erano praticamente sinonimi[5]. Quando l'IBM introdusse il PC, per il grande pubblico americano fu come se il PC nascesse in quel momento. I grandi utenti, quali banche, industrie, apparati statali ecc., fino ad allora refrattari all'uso dei PC, trovarono del tutto naturale il loro impiego, anche in virtù della supposta possibilità di integrazione con gli esistenti *mainframe* IBM. La rivista *Time*, che tradizionalmente dedica la copertina dell'ultimo numero di ogni annata al personaggio maggiormente distintosi sul pianeta nel corso dell'anno, nel 1981, con sorpresa dei lettori, dedicò la copertina al PC invece che a una persona.

A titolo di curiosità, vale la pena di ricordare che l'IBM non aveva posto particolare cura o determinazione nel produrre il suo PC. Prova ne sia il fatto che esso venne progettato e prodotto in una sede periferica, di minor importanza, situata in Florida, e cioè lontano da quella vasta area nel nord dello stato di New York, dove la società aveva i principali centri di progettazione e produzione. Il PC IBM presentava solo due unità a disco flessibile (*floppy*) da $5''\frac{1}{4}$. Il disco rigido, da 10 MB, comparve qualche tempo dopo sulla versione PC/XT. Retaggio del precedente mondo dei personal computer a basso costo, sul retro del PC IBM faceva bella mostra di sé un connettore per il collegamento delle unità di registrazione/lettura di cassette audio, allora molto in voga tra chi usava i microcalcolatori.

Ci sarebbe da fare anche un po' di storia sul DOS. Sono state fatte diverse ricostruzioni sul motivo che ha determinato il successo del DOS. Una delle più accreditate è quella che segue. L'IBM voleva una versione *propria* del CP/M, con caratteristiche che la distinguessero rispetto al resto dei produttori. Sembra che il produttore del CP/M (la società Digital Research), forte della sua posizione di predominanza sul mercato dei sistemi operativi per personal computer, abbia tenuto una posizione alquanto "distaccata" nei confronti dell'IBM, spingendo i responsabili del colosso informatico a rivolgersi anche a una piccola ditta, la Microsoft, nota per un diffuso interprete BASIC per i micro a 8 bit. Per la Digital Research questo era un progetto, per Microsoft, questo era *il progetto* e si mise di buon grado a disposizione del gigante dell'informatica. Iniziava così il percorso che avrebbe portato la Microsoft a contendere il primato alla stessa IBM e fare del suo presidente Bill Gates, allora poco più che un ragazzino, il Paperon de' Paperoni dei nostri giorni.

[4]Si tenga presente che la caratteristica che distingueva in modo apprezzabile l'8088 da 8085, Z80, 6800 e da tutti gli altri microprocessori a 8 bit della precedente generazione, consisteva proprio nel fatto che questi ultimi avevano uno spazio di indirizzamento limitato a 64 KB. L'8088 era il primo micro a 8 bit a sfondare tale confine e a mettere a disposizione uno spazio di memoria – allora ritenuto spropositatamente ampio, tanto che il software di base del PC ne impiegò ben un terzo come memoria ROM.

[5]Non è il solo caso di identificazione di un marchio di fabbrica con un tipo di prodotto merceologico. In Italia, la parola "Ferodo" viene usata per indicare il materiale impiegato nei freni a ganascia degli autoveicoli, dal nome suo iniziale e principale produttore. Chi ha visto il celebre film di Stanley Kubrick "2001 Odissea nello spazio" ricorderà che il calcolatore responsabile della gestione del controllo la navicella spaziale, si chiamava HAL 9000. La sigla HAL era una trasposizione di IBM; si ottiene infatti prendendo ordinatamente le lettere che nell'ordine alfabetico precedono quelle in IBM.

Avanzamento e retrocompatibilità
Come abbiamo accennato sopra, il vantaggio temporale iniziale, l'attenzione al software e l'adozione da parte di IBM favorirono i processori Intel rispetto ai competitori. AMD stipulò un accordo con Intel per lo sviluppo e la produzione della CPU 8086 (successivamente le strade delle due società si sono separate, non senza qualche controversia giudiziaria). Il PC IBM-compatibile diventò lo standard. E crebbe il volume di software sviluppato per esso.

La scelta vincente di Intel e AMD è stata quella della compatiblità all'indietro. Ovvero, ogni nuovo processore doveva essere in grado di eseguire i programmi sviluppati per i modelli precedenti, in modo da salvaguardare l'enorme patrimonio software che andava accumulandosi. L'affermarsi delle architetture RISC non modificò questa linea, anche se alcuni concetti affermatisi con i RISC vennero recepiti a livello di microarchitettura. Ad esempio, la trasformazione interna delle istruzioni architetturali in istruzioni interne stile RISC (denominate micro-operazioni), per una più efficiente esecuzione. Il programmatore, fatti salvi i miglioramenti da modello a modello, ha sempre continuato a vedere il medesimo modello di programmazione.

A riprova di quanto sia importante sul mercato la compatibilità all'indietro, basterà accennare a cosa è successo nel passare ai modelli a 64 bit. A fine anni novanta, Intel, in collaborazione con Hewlett-Packard, si era imbarcata nella definizione di una nuova architettura denominata IA-64, ovvero nello sviluppo dei processori Itanium di cui si parla nell'Appendice D. Questi avrebbero dovuto essere retrocompatibili –per simulazione– con il software ×86[6]. Nel 2003, AMD ha introdotto il primo processore a 64 bit compatibile all'indietro con la famiglia ×86. La compatibilità è stata ottenuta direttamente in hardware, cioè in modo molto più efficiente che per simulazione. La risposta positiva del mercato ha indotto Intel a seguire la strada di AMD, lasciando sostanzialmente al suo destino la linea Itanium, prontamente abbandonata dai grandi nomi del software. Si noti che l'architettura a 64 bit ha semplificato l'architettura tradizionale, il cui tratto più caratteristico, la memoria segmentata, è stato sostanzialmente perso. Ma la compatibilità con la precedente architettura, consentendo una transizione morbida verso i 64 bit, ha subito orientato le scelte dei costruttori di calcolatori.

Dal 2006 anche la Apple ha iniziato ad adottare processori Intel, abbandonando la linea PowerPC che essa stessa aveva contribuito a definire, proprio in ottica antagonistica rispetto alla ×86.

Nel seguito tratteremo piuttosto a fondo l'8086, per mostrare gli aspetti fondamentali dell'architettura e in considerazione della sua (relativa) semplicità. Successivamente si passerà a esporre l'architettura a 32 bit. La parte finale sarà dedicata alla versione a 64 bit. Buona parte dei processori presi in considerazione in questa appendie appartengono al passato, perciò nell'illustrarli occorrerebbe usare il tempo imperfetto. Ma di norma si preferisce usare il tempo presente, come se essi fossero correntemente in uso.

C.2 Le basi dell'architettura: il micro 8086

La CPU 8086, fece la sua apparizione nel Giugno 1978 (Tabelle 1.1 e 1.4 nel capitolo di introduzione del libro). Lo stato della tecnologia del tempo, i vincoli imposti alla

[6] I titolo "Intel®64 and IA-32 Architectures" compare sulla prima pagina dei manuali dal 2007, in precedenza erano intitolati "IA-32 Intel®Architecture". Chi allora avesse fatto una ricerca con Google per "64 bit architecture" sarebbe approdato esclusivamente a pagine relative all'architettura Itanium.

progettazione (parvenza di compatibilità con il microprocessore 8085 a 8 bit e l'impiego di un integrato a 40 piedini) e l'intento di definire un sofisticato sistema di protezione hanno avuto una notevole influenza nel determinarne l'architettura e nel renderla non poco complicata.

L'8086 è un processore a 16 bit e spazio di indirizzamento di 1 MB. Il numero medio di cicli di clock per istruzione C_{PI} può essere valutato pari a 15. A 5 MHz (frequenza di introduzione) il tempo medio per eseguire un'istruzione è: $15 \cdot 200$ ns = 3 μs, corrispondente a un livello di prestazioni di circa 0,33 MIPS.

C.2.1 Il modello di programmazione

Il modello di programmazione dell'8086 si compone dei registri riportati in Figura C.1. I registri possono essere così raggruppati:

- otto registri di uso generale, di cui:
  – quattro (AX, BX, CX e DX) aventi funzione di registri dati,
  – quattro (SP, BP, SI e DI) usati come puntatori o registri indice;
- quattro registri di segmento (CS, DS, SS e ES);
- un registro (IP) usato come puntatore di istruzione in congiunzione con CS;
- un registro di stato, che l'Intel chiama registro dei Flag.

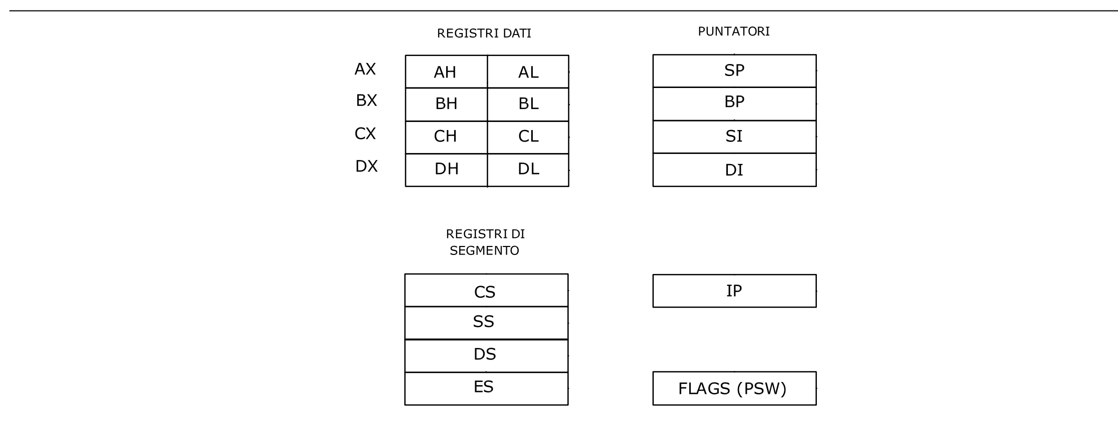


Figura C.1 Modello di programmazione 8086/88.

I registri dati

Come indicato in Figura C.1, i registri dati possono essere impiegati sia come registri da 16 bit, sia da 8.

Come registri a 16 bit, vengono individuati con i termini AX, BX, CX, DX mentre con le denominazioni AH, BH, CH, DH e AL, BL, CL, DL si identificano rispettivamente i byte più e meno significativi dei registri a 16 bit, che possono essere usati individualmente come registri a 8 bit[7]. Nella maggior parte dei casi, questi registri possono essere impiegati

[7] Nella pubblicistica Intel del primo periodo di vita dell'8086 veniva posto un forte accento sul fatto che i registri dati rappresentavano un superinsieme dei registri della precedente famiglia a 8 bit (8080/8085).

senza alcuna distinzione in operazioni aritmetiche e logiche. Tuttavia essi hanno anche ruoli specifici.

- AX ha funzione tipica di *accumulatore*. Può esser usato in tutte le istruzioni di I/O, nelle istruzioni relative alle stringhe e nelle operazioni aritmetiche. Un numero ristretto di istruzioni richiede obbligatoriamente AX.
- BX può essere usato anche come registro base per il calcolo degli indirizzi; per questo motivo viene indicato come *base register*.
- CX viene usato anche come contatore in certe istruzioni; per questo motivo viene indicato come *count register*.
- DX viene designato come *data register*. È richiesto da alcune operazioni di ingresso/uscita, come pure dalle operazioni di moltiplicazione e divisione che, coinvolgendo grandi valori, presuppongono la coppia DX, AX.

I registri indice e i puntatori

Questi registri di solito contengono gli scostamenti all'interno dei segmenti. SP e BP hanno prevalente funzione di puntatori, mentre SI e DI vengono usualmente usati come indici.

- SP (*Stack Pointer*): è il puntatore alla cima dello stack.
- BP (*Base Pointer*): viene principalmente usato come puntatore entro lo stack, ma può anche essere impiegato come generico registro indice.
- SI (*Source Index*): registro indice di uso generico. Il nome deriva dal fatto che certe istruzioni (di stringa) richiedono che la stringa sorgente sia necessariamente individuata tramite SI.
- DI (*Destination Index*): registro indice di uso generico. Il nome deriva dal fatto che, in certe istruzioni che manipolano stringhe, la stringa di destinazione deve essere necessariamente individuata tramite DI.

I registri di segmento

L'aspetto più caratteristico della CPU 8086 è la segmentazione della memoria, di cui si parla al successivo Paragrafo C.2.2. I registri di segmento vengono impiegati proprio per tenere traccia della posizione in memoria dei segmenti correntemente in uso.

L'impiego usuale dei quattro registri è questo: CS identifica il segmento di codice (*code segment*) corrente, DS il segmento di dati (*data segment*) corrente, SS il segmento di stack (*stack segment*) corrente ed ES il segmento extra (*extra segment*) corrente.

Il registro IP

I codici di istruzione vengono sempre prelevati dal segmento di codice. A questo proposito è necessario un registro che contenga l'offset dell'istruzione successiva da eseguire, riferito al segmento di codice corrente. IP contiene la posizione dell'istruzione riferita alla base del segmento di codice.

Il registro di stato (FLAGS)

Il registro di stato dell'8086 contiene 9 indicatori di 1 bit, detti anche *flag*. Di questi, 6 registrano informazioni sullo stato del processore (flag di stato) e 3 servono a controllare le operazioni del processore (flag di controllo). In Figura C.2 viene schematizzato il formato del registro di stato.

I flag di stato sono i seguenti.

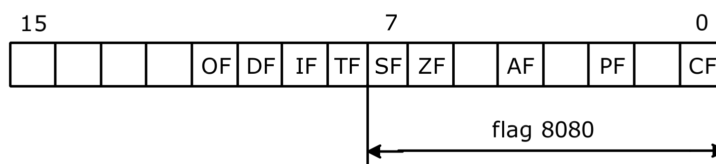


Figura C.2 Formato della parola di stato. Il byte meno significativo coincide con la parola di stato dei precedenti microprocessori Intel a 8 bit (8080 e 8085).

- CF (*Carry Flag*): indica se un'istruzione ha generato un riporto dal bit più significativo.
- AF (*Auxiliary Flag*): indica se c'è un riporto dal bit 3 durante un'addizione o un prestito dal bit 3 durante una sottrazione. È usato in aritmetica in BCD.
- OF (*Overflow Flag*): indica se un'istruzione ha generato un trabocco.
- SF (*Sign Flag*): indica se un'operazione ha generato un risultato negativo.
- ZF (*Zero Flag*): indica se un'istruzione ha generato zero come risultato.
- PF (*Parity Flag*): indica se un'istruzione ha generato un risultato con un numero pari di bit unitari.

I flag di controllo sono i seguenti.

- DF (*Direction Flag*): serve a controllare la direzione di manipolazione delle stringhe da parte di alcune istruzioni a esse riservate. DF indica se la stringa viene letta/scritta a partire dall'elemento che ha indirizzo più basso o da quello di indirizzo più alto.
- IF (*Interrupt Flag*): abilita o disabilita le interruzioni esterne.
- TF (*Trap Flag*): permette l'esecuzione di un'istruzione per volta (*single step*); è fondamentale per il *debugging*[8].

C.2.2 Organizzazione della memoria

L'organizzazione della memoria (ispirata a quella del sistema operativo Multics[9]) è il tratto più caratteristico del micro 8086 ed ha condizionato pesantemente l'evoluzione dell'architettura. La CPU 8086 può indirizzare fino a un massimo di 1 MB di memoria. Questo spazio è logicamente suddiviso in segmenti.

La segmentazione

Un segmento è un'unità logica di memoria che può avere una estensione massima di $2^{16} = 64\,\text{KB}$. Ogni segmento si compone di locazioni di memoria contigue e costituisce una unità di memoria indirizzabile separatamente e indipendentemente da altri segmenti. Un segmento può essere sistemato in qualunque posizione della memoria fisica, purché inizi da una locazione avente un indirizzo divisibile esattamente per 16 (un gruppo di 16 byte che inizia a un indirizzo multiplo di 16 viene detto *paragrafo*). A parte l'indirizzo di partenza, non ci sono altre limitazioni. I segmenti possono essere adiacenti, disgiunti,

[8] Il termine si riferisce figuratamente all'eliminazione dai programmi di errori di basso livello, *bugs* o *bachi* in gergo informatico.

[9] Multics sta per *Multiplexed Information and Computing Service*. Venne sviluppato a partire dal 1964 presso il MIT (*Massachusetts Institute of Technology*). Esso ha avuto grande influenza sui sistemi operativi moderni. Uno dei sui aspetti caratteristici era la protezione a più livelli.

parzialmente o totalmente sovrapposti. Conseguentemente, una locazione di memoria fisica può interessare uno o più segmenti.

Al Paragrafo C.2.1, si è visto che in CPU ci sono i 4 registri di segmento CS, DS, SS e ES: in un dato momento essi contengono l'indirizzo di partenza (la base) dei segmenti correnti. Cambiando il contenuto di questi registri, dal medesimo programma si possono indirizzare differenti segmenti. Conseguentemente gli indirizzi generati dai programmi hanno due componenti: un registro di segmento e lo scostamento entro il segmento e si rappresentano come SR:OFFSET (si faccia riferimento al Paragrafo 3.5.1 del libro). Gli indirizzi in questa forma si denotano come *indirizzi logici*.

Il modo in cui vengono calcolati gli indirizzi fisici dagli indirizzi logici è stato illustrato in Figura 7.15 del libro. I registri di segmento contengono la base del segmento, ovvero l'indirizzo di partenza diviso per 16. Questa tecnica di costruzione dell'indirizzo fisico da quello logico ha consentito di indirizzare uno spazio di 1 MB di memoria con registri di segmento di soli 16 bit anziché 20. Solo l'uscita del sommatore, appoggiata sul registro che interfaccia il bus degli indirizzi, è di 20 bit. All'epoca in cui uscì l'8086, la densità di integrazione era lontana dai livelli correnti. Risparmiare sulla dimensione dei registri rendeva disponibile per altri scopi parte dell'area di silicio del chip.

I segmenti del codice, dei dati e dello stack formano tre diversi spazi logici associabili in modo naturale all'esecuzione dei programmi. Questa suddivisione porta a una concettualizzazione dei programmi, secondo la quale c'è una netta divisione tra codice, dati e dati temporanei[10].

In Figura C.3 viene data una rappresentazione di una possibile allocazione dei segmenti. Nel caso specifico, i segmenti sono tenuti separati, ma niente toglie che due o più di essi possano sovrapporsi, parzialmente o totalmente. In figura si indica tutto lo spazio occupabile dal segmento; nella pratica un segmento sarà limitato a quanto necessario. Ovviamente, i programmi di dimensione superiore a 64 KB, devono essere organizzati in più segmenti e i relativi registri di segmento dovranno essere modificati per passare da un segmento all'altro. Per quanto si riferisce al codice, possono essere usate le istruzioni di salto *intersegmento*.

In generale la segmentazione ha le seguenti caratteristiche.

- Facilita l'uso di aree separate per il codice, i dati e lo stack; in tal modo l'organizzazione del programma in memoria riflette la struttura logica del programma stesso. Per contro, questa suddivisione tende a oscurare la leggibilità del programma rispetto a uno spazio di indirizzi piatto.
- Permette una facile rilocazione dei programmi in memoria; infatti, essendo gli indirizzi fisici ottenuti come somma di uno scostamento rispetto al contenuto di un registro, la rilocazione richiede il solo aggiustamento del contenuto dei registri di segmento.

Uso dei registri di segmento

I registri di segmento hanno una funzione predefinita, ma c'è la possibilità di alcuni impieghi alternativi, come illustrato in Tabella C.1.

La colonna "Registro Normale" indica il registro di segmento che viene normalmente codificato nell'istruzione come base del segmento. Per esempio, l'istruzione in linguaggio assembler:

[10] Il quarto segmento ha solo una giustificazione pratica: esso veniva ottenuto al solo costo del relativo registro ES. Esso viene utilizzato per scopi specifici; peraltro, in certe situazioni risulta utile disporre di un segmento di appoggio.

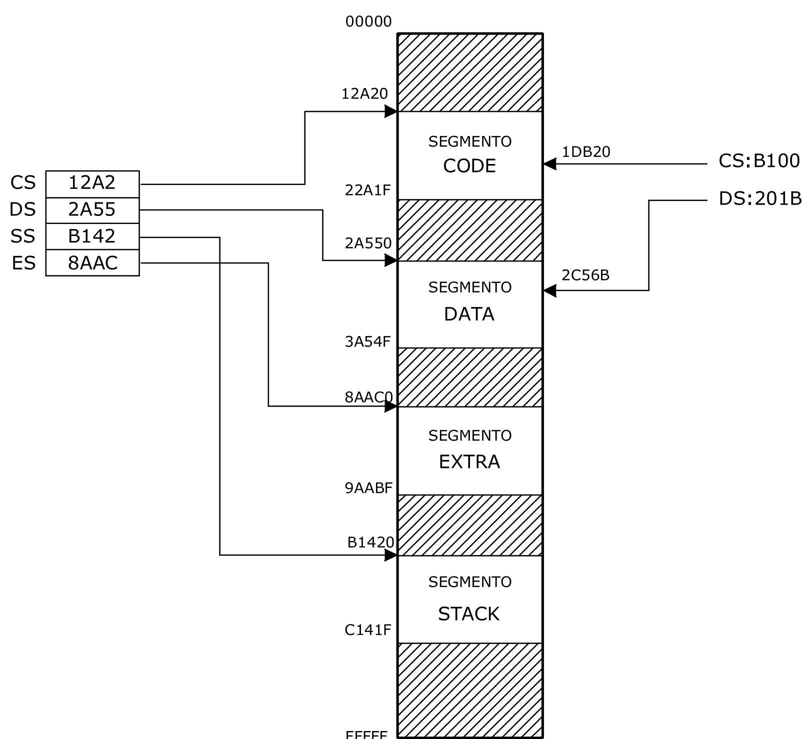


Figura C.3 Esempio di sistemazione dei quattro segmenti in memoria.

Tipo di riferimento	Registro Normale	Registro Alternativo	Scostamento
Fetch	CS	–	IP
Operazioni di stack	SS	–	SP
Variabili (eccetto quanto segue)	DS	CS, ES, SS	EA
Stringhe sorgenti	DS	CS, ES, SS	SI
Stringhe destinazione	ES	–	DI
BP (usato come base)	SS	CS, DS, ES	EA

Tabella C.1 Impiego dei registri di segmento. La coppia (Registro di segmento:Scostamento) costituisce l'indirizzo logico. Le istruzioni vengono sempre prelevate dal segmento di codice, mentre IP fornisce lo scostamento. Allo stesso modo, le operazioni che operano sullo stack (PUSH e POP) usano sempre SS come base e SP come scostamento. In tabella, con EA si denota l'indirizzo effettivo (*Effective Address*), ovvero il valore dell'indirizzo calcolato dalla CPU, a monte del registro di segmento. Si veda il Paragrafo C.2.5.

```
MOV AX,VAR      ;AX ← M[DS:Offset(VAR)];
```

dove `VAR` è il nome assegnato a una variabile o costante definita nel segmento dei dati, genera un codice che ha l'effetto di caricare in AX il contenuto della locazione di memoria scostata di `Offset(VAR)` dalla base del segmento individuato da DS.

Per impiegare un registro alternativo in luogo di DS, il programmatore deve esplicitamente indicarlo. Per esempio, volendo usare ES, si dovrà scrivere:

```
ADD AX,ES:VAR   ;AX ← AX+M[ES:offset(VAR)]
```

In questo caso l'assemblatore genera per l'istruzione il medesimo codice generato per la precedente, ma lo fa precedere da un byte contenente il prefisso di segmento (*segment override prefix*, Paragrafo 3.4.2) che al tempo di esecuzione determina l'impiego di ES in luogo di DS.

Allineamento

L'organizzazione della memoria dell'8086 è del tipo *little endian*. La CPU può leggere/scrivere 8 o 16 bit, Nel caso di lettura/scrittura a 16 bit, non è richiesto l'allineamento. Ovviamente l'8088 legge sempre 8 bit e la lettura/scrittura di 16 bit richiede comunque due cicli di bus.

Lo stack in memoria

Il microprocessore gestisce lo stack in memoria attraverso i registri SS e SP. Ovviamente, un dato programma può prevedere un numero qualsivoglia di stack, anche se, a un preciso istante di tempo, è attivo solo lo stack corrente, individuato tramite il registro di segmento SS.

Lo stack ha ampiezza di parola. Sono previste due operazioni: PUSH e POP. La prima inserisce sulla testa dello stack 16 bit, la seconda li preleva; SP viene aggiornato in modo conseguente.

Mentre SS contiene la base dello stack corrente, SP contiene il puntatore alla cima, o testa; in altri termini, SP contiene lo scostamento della cima dello stack rispetto a SS. Si noti però che, mentre SP dà la cima dello stack, SS non ne dà il *fondo*; infatti lo stack si sviluppa dagli indirizzi alti verso i bassi. In pratica, il valore iniziale di SP è il valore più grande dello scostamento della cima dello stack. La Figura C.4 illustra il funzionamento dello stack.

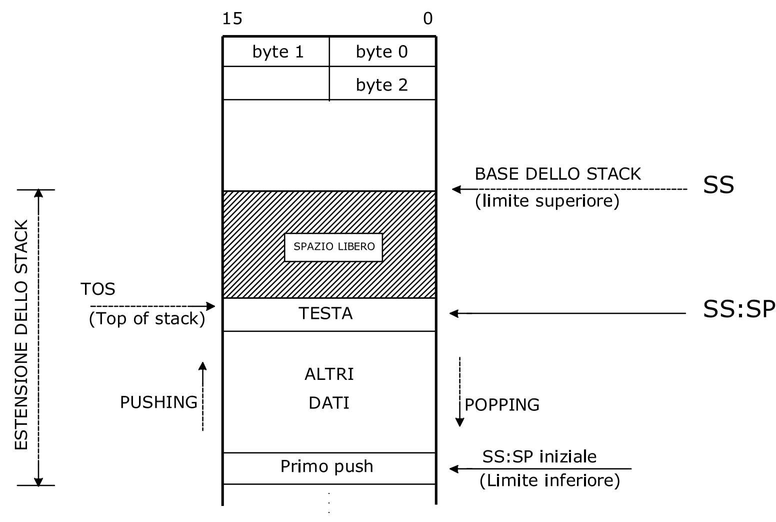


Figura C.4 Lo stack dell'8086.

Al fine di semplificare l'uso dello stack, è comunque prevista la possibilità di accedervi in via diretta. A tale scopo serve il registro BP. Per esempio:

```
MOV AX, [BP] ;AX ← M[SS:BP]
```

Per l'impiego dei registri BP e SP si faccia riferimento al Paragrafo 3.7.1 del libro.

C.2.3 La fase di partenza

Quando la CPU viene messa sotto tensione, ovvero sul *reset*, a CS viene assegnato il valore FFFF mentre a IP viene assegnato il valore 0. Conseguentemente la prima istruzione eseguita è quella alla locazione FFFF0. Essendoci pochissime posizioni tra questa posizione e il confine estremo della memoria[11] è necessario che essa contenga un salto alla prima effettiva istruzione del programma. È perciò obbligatorio che la parte alta dello spazio degli indirizzi sia coperto da memoria di tipo ROM, allo scopo di garantire che all'atto della messa sotto tensione entri in scena il *firmware* di inizializzazione del sistema. Si tratta della parte più interna del sistema operativo (il BIOS, da *Basic Input/Output System*).

C.2.4 Il repertorio delle istruzioni

L'8086 ha un repertorio di istruzioni molto ampio, come di solito hanno le macchine microprogrammate. A livello di codice assembler l'8086 ha un repertorio di circa 100 istruzioni, ma a ciascun codice assembler può corrispondere più di una istruzione di macchina. Per esempio, l'istruzione `MOV`, usata per lo spostamento di dati, viene impiegata per trasferire dati da e verso registri e memoria, a 8 o a 16 bit. Allo mnemonico `MOV` del linguaggio Assembler corrispondono 28 istruzioni di macchina (*move word register to memory, move byte immediate to register* ecc.). L'assemblatore traduce nel codice opportuno l'istruzione assembler. Naturalmente questa varietà di istruzioni macchina comporta una struttura dei codici e dei formati alquanto complicata (Paragrafo 3.4.2 del libro).

In Tabella C.2 vengono riportate le forme mnemoniche di alcune istruzioni raggruppate per classi.

La CPU non dispone dell'unità per le operazioni in virgola mobile, ma è previsto il coprocessore 8087, un dispositivo esterno che estende il repertorio di istruzioni aggiungendo un insieme completo di istruzioni per operazioni in virgola mobile. Di esso si parla al Paragrafo C.2.6, pag. 14.

C.2.5 Modalità di indirizzamento

A costo di annoiare il lettore, invitato comunque a consultare i manuali e la vasta letteratura Intel, vale la pena di spendere due parole sulle modalità di indirizzamento.

Indirizzamento dei registri di CPU e degli operandi immediati

Le istruzioni che usano come operandi solo i registri di CPU sono molto compatte, in quanto la codifica dei registri richiede pochi bit. Esempio:

```
MOV AH,BL        ; AH ← BL
```

La macchina prevede anche la possibilità di specificare gli operandi immediati. In questo caso, l'operando diventa parte del codice della istruzione e l'esecuzione dell'istruzione non richiede un ulteriore ciclo di lettura dalla memoria, in quanto il dato è presente in CPU per effetto del fetch dell'istruzione[12].
Esempio:

```
MOV AX,2475      ; AX ← 2475
```

Indirizzamento in memoria

Esiste una varietà molto ampia di modi di indirizzamento. In Figura C.5 viene data una schematizzazione che li riassume tutti. Si noti che nel calcolo di EA (*Effective Address*) possono

---

[11] Peraltro, parte delle posizioni dell'ultimo paragrafo di memoria sono riservate a usi speciali.

[12] Ovviamente un operando immediato può essere solo di lettura.

Trasferimento dati	
MOV	Sposta un byte o una parola
PUSH	Impila una parola sullo stack
OUT	Trasferisce un byte o una parola in uscita
LEA	*Load Effective Address*
POPF	Estrae la parola di stato dallo stack
Istruzioni aritmetiche	
ADD	Somma byte o parole
CMP	Compara byte o parole
IMUL	Moltiplica (interi) byte o parole
Manipolazione dei bit	
AND	Esegue l'AND tra due byte o due parole
SHL	Scorrimento a sinistra
RCR	Ruota a destra attraverso il riporto
Manipolazione stringhe	
MOVSB	Sposta una stringa di byte
CMPS	Confronta stringhe
Salti	
JMP	Salto incondizionato
CALL	Chiamata di sottoprogramma
JE	Salta se uguale
JNGE	Salta se non maggiore o uguale
INT	Interruzione (software)
IRET	Ritorno da (routine) interruzione
Controllo della CPU	
STC	Porta a 1 il bit di riporto
CLI	Disabilita il sistema di interruzione
STD	Stabilisce la direzione di scansione stringhe
Sincronizzazione con l'esterno	
WAIT	Passa allo stato di *wait*
Istruzione di non operazione	
NOP	Non fa niente

Tabella C.2 Alcune istruzioni del repertorio divise per classe. A certi mnemonici di istruzione corrispondono più codifiche di macchina. Tanto per esemplificare l'istruzione `MOV` può riguardare il trasferimento, da registro a registro, da memoria a registro, da registro a memoria, da immediato a registro, da immediato a memoria. A seconda di cosa compare, l'assemblatore trasferisce 8 o 16 bit. Per esempio `MOV al,var` trasferisce un byte, mentre `MOV AX,var` trasferisce una parola (16 bit); un'istruzione come `MOV AX,BL` non è consentita per la diversa misura tra sorgente e destinazione.

concorrere fino a tre componenti: una base, un indice e uno scostamento. Nel caso di sola base questa deve essere presa da uno di questi 4 registri BX, BP, SI e DI; nel caso di base e indice la base deve essere BX o BP, mentre l'indice deve essere SI o DI. Diamo ora alcuni esempi.

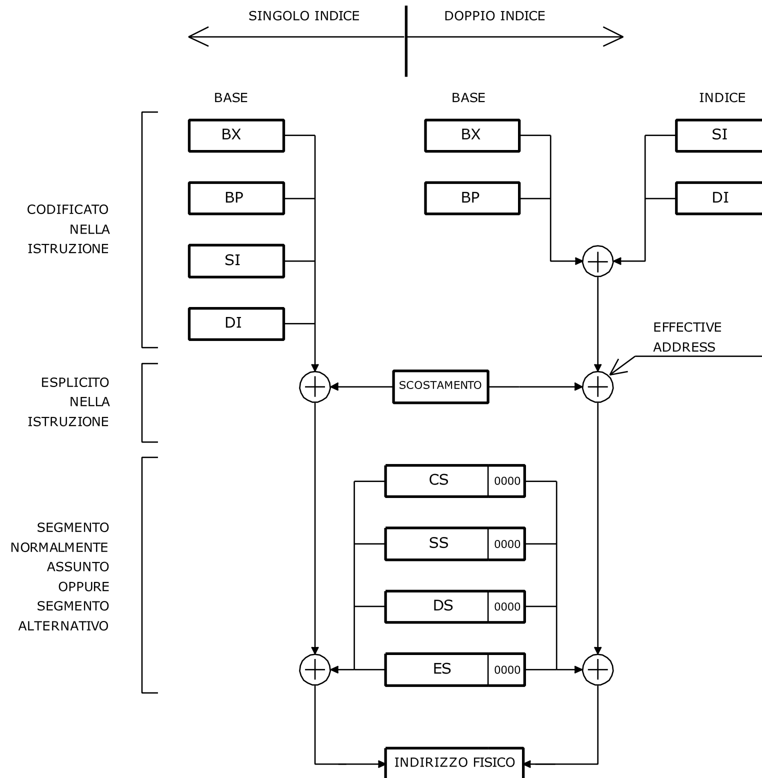


Figura C.5 Metodo di calcolo dell'indirizzo di memoria. Si noti che lo scostamento può essere, a seconda dei casi, di 8 o di 16 bit. L'indirizzo effettivo *Effective Address (EA)* è quello calcolato a monte del contributo del registro di segmento, in base al contenuto dell'istruzione. L'indirizzo fisico (lineare) è ottenuto sommando ad EA l'indirizzo di partenza del segmento, cioè il contenuto del registro di segmento moltiplicato per 16.

- *Indirizzamento diretto*. Quando non interviene alcun registro. Esempio:
  `MOV AX,VAR      ;AX ← M[Offset(VAR)]`
  EA è dato dallo `SCOSTAMENTO` contenuto nel campo dell'istruzione (Figura C.5), indicato come `Offset(VAR)`. In realtà l'effetto dell'istruzione è
  `AX ← M[DS:Offset(VAR)]` in quanto l'indirizzo fisico (lineare) deriva sempre da quello logico (segmentato).
  Qui e nel seguito si evita di indicare il registro di segmento implicato. A tale scopo si faccia riferimento alla Tabella C.1.
- *Indirizzamento indiretto attraverso un registro*. Quando l'istruzione non presenta il campo dello scostamento e viene specificata solo la base. Esempio:
  `MOV [BX],AL      ;M[BX] ← AL`
- *Indirizzamento relativo a un registro*. EA è la somma dello scostamento codificato nell'istruzione e del contenuto del registro specificato. Questo è il caso del percorso di sinistra di Figura C.5. Esempio:

```
MOV AX,VAR[BX]      ; AX ← M[Offset(VAR)+BX]
```

- *Indirizzamento indiretto attraverso un registro e indiciato.* EA è la somma del contenuto dei due registri. Esempio:

```
MOV [BX][DI],AX      ; M[BX+DI] ← AX
```

- *Indirizzamento relativo a un registro e indiciato.* Corrisponde al caso precedente, con l'aggiunta dello scostamento codificato nell'istruzione. Questo è il caso del percorso di destra di Figura C.5. Esempio:

```
MOV CX,VAR[BX][SI]     ; CX ← M[Offset(VAR)+BX+SI]
```

Indirizzamento delle porte di I/O

Ci si riferisce all'indirizzamento previsto dalle due operazioni di Ingresso/Uscita (IN e OUT) previste nel set di istruzioni[13].

Ci sono due modalità.

- L'indirizzo assegnato alla porta è codificato nell'istruzione. A tale scopo viene impiegato un campo di 8 bit. Ne consegue la possibilità di indirizzare 256 porte di ingresso e 256 porte di uscita. Esempio:

```
IN AL,PORTA      ; AL ← PORTA di Ingresso
```

- L'indirizzo della porta è contenuto nel registro DX. In questo caso lo spazio di I/O indirizzabile si estende fino a raggiungere 64 KB di ingresso e 64 KB di uscita. Esempio:

```
OUT DX,AH        ; Porta DX ← AH
```

Indirizzamento nei salti

I salti sono di due tipi: intrasegmento e extrasegmento. Nel primo caso viene modificato solo IP; nel secondo vengono modificati CS e IP. Si hanno le seguenti possibilità.

- intrasegmento diretto: EA del salto è la somma dello scostamento e del contenuto corrente di IP. Lo scostamento può occupare 16 o 8 bit. Nel secondo caso, si parla di salto corto. Le istruzioni di salto condizionato codificano solo scostamenti di 8 bit.
- intrasegmento indiretto: EA del salto è il contenuto di un registro o di una parola di memoria a cui si accede usando uno qualunque dei modi di indirizzamento visti per gli operandi (ad eccezione del modo immediato).
- extrasegmento diretto: sostituisce il contenuto di IP con una parte dell'istruzione e il contenuto di CS con un'altra parte dell'istruzione. Lo scopo di questo modo di indirizzamento è di provvedere al salto da un segmento all'altro.
- extrasegmento indiretto: sostituisce il contenuto di IP e di CS con il contenuto di due parole consecutive nella memoria, a cui ci si riferisce usando uno qualunque dei modi di indirizzamento (ad eccezione di quello immediato).

C.2.6 Il coprocessore aritmetico

L'8086 conteneva "solo" 29.000 transistori. Nella seconda metà degli anni settanta del secolo scorso, le tecnologie di integrazione erano estremamente limitate rispetto alle attuali. Non fu possibile integrare la logica per l'aritmetica in virgola mobile. Però i progettisti previdero di utilizzare un *coprocessore* realizzato come integrato a parte.

Un coprocessore è un dispositivo che espande "in modo trasparente" le funzionalità del processore di base, tale che per il programmatore l'accoppiata CPU-coprocessore appare come un unico dispositivo — in grado di eseguire operazioni non disponibili per la sola CPU. Il fatto di realizzare la logica del coprocessore come integrato a parte comporta la necessità di definire un protocollo con il quale i due interagiscono.

[13] Ovviamente, se le porte di I/O vengono mappate in memoria, valgono tutte le modalità previste per l'indirizzamento in memoria.

Il coprocessore, denominato 8087, ovvero NPX (*Numerical Processor eXtension*,), venne introdotto nel 1980, due anni dopo l'8086. All'epoca era il dispositivo commerciale a maggior livello di integrazione (circa 750.000 transistori); i primi prototipi immessi sul mercato tolleravano poco l'alzarsi della temperatura di funzionamento. L'8087 migliorava di circa 100 volte i tempi delle operazioni in virgola mobile, che, in sua assenza, dovevano essere eseguite per emulazione da programma usando le istruzioni tra interi disponibili. Esso veniva montato direttamente sul bus della CPU, con la quale scambiava alcuni segnali di controllo ai fini del coordinamento delle operazioni.

Dal modello 80486 le funzionalità dell'8087 sono integrate nello stesso chip della CPU. Anche per questa ragione non staremo qui a descrivere le caratteristiche del coprocessore 8087. Tuttavia, ritenendo istruttivo illustrare il modo in cui i coprocessori interagiscono con la CPU, nell'Appendice E è stato previsto un paragrafo che tratta la questione, sebbene in riferimento all'Architettura ARM, per la quale l'uso dei coprocessori è un aspetto importante. Il tale illustrazione vengono fatti alcuni accenni al coprocessore 8087.

C.3 Architettura a 32 bit

Il passaggio a 32 bit si è avuto con l'introduzione dell'80386 nell'Ottobre 1985. Oltre al parallelismo raddoppiato, il 386 introduceva la paginazione della memoria virtuale a valle della segmentazione (la memoria virtuale segmentata era stata introdotta in precedenza con l'80286). La serie dei processori a 32 bit che ha fatto seguito al 386 è stata denominata da Intel IA-32. In questa parte si richiamano gli aspetti fondamentali di questa architettura.

C.3.1 Modalità di funzionamento a 32 bit

L'architettura IA-32 prevede 4 modalità di funzionamento, come illustrato in Figura C.6.

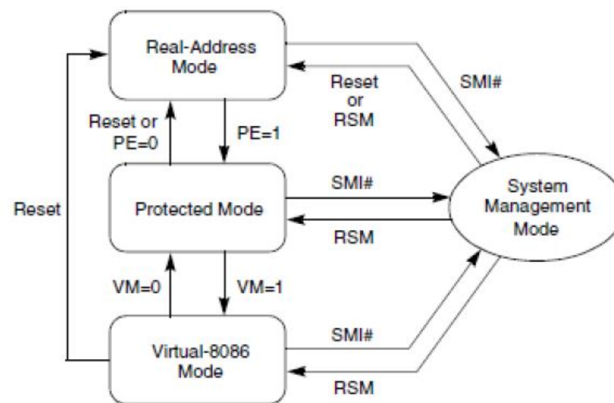


Figura C.6 Stati di funzionamento delle macchine a 32 bit. La figura è riportata senza modifiche dai manuali Intel. Il simbolo "#" indica negazione del segnale. PE indica un bit del registro di controllo CR0, esso determina il passaggio tra la modalità protetta e la modalità con indirizzi reali. VM è un bit del registro EFLAGS e determina il passaggio tra la modalità protetta e la modalità 8086 virtuale. SMI# indica l'interruzione che si generi sul piedino omonimo; RSM indica l'esecuzione dell'istruzione RSM che fa abbandonare lo stato "System management".

Il significato delle 4 modalità è questo:

Modalità Protetta. Corrisponde allo stato *nativo*, ovvero allo stato in cui la macchina opera naturalmente. In questa modalità è attiva la gestione della memoria virtuale.

Modalità con indirizzi reali (*Real address mode*). In questa modalità la CPU si comporta come un 8086. Alla partenza o sul reset la CPU viene messa in questo stato (si veda il Paragrafo C.2.3, pag. 11). Il passaggio alla modalità protetta deve essere comandato esplicitamente portando a 1 il bit PE del registro di controllo CR0.

Modalità 8086 virtuale. A partire dal modo protetto, la CPU può essere portata in modalità virtuale 8086 dove si comporta come un 8086 (indirizzamento reale) ma esegue come un *task* indipendente. Questo permette da un lato di avere un sistema *multitasking* in cui ogni processo esegue un programma 8086, dall'altro di far funzionare questi programmi in modalità protetta, con tutti i vantaggi che da essa conseguono.

Modalità gestione del sistema (*System management mode*, SMM). A questa modalità si passa quando viene asserita la speciale richiesta di interruzione sul piedino SMI del processore. Nello stato SMM il processore vede uno spazio degli indirizzi separato da quello normale, ma mantiene immutato lo stato dei programmi o processi correnti. Al ritorno al funzionamento normale (tramite l'istruzione RSM) la CPU viene riportata allo stato che aveva prima dell'interruzione. Lo stato SMM serve a implementare funzioni speciali o di ausilio durante le fasi di sviluppo.

Nel seguito non ci occuperemo più della modalità SMM. Inoltre, gli argomenti esposti si riferiranno sostanzialmente alla modalità protetta.

C.3.2 Il modello di programmazione a 32 bit

In Figura C.7 viene mostrato il modello di programmazione del 386. Esso costituisce il modello di base dell'architettura $\times 86 - 32$. I registri di CPU sono stati portati a 32, con l'eccezione dei re-

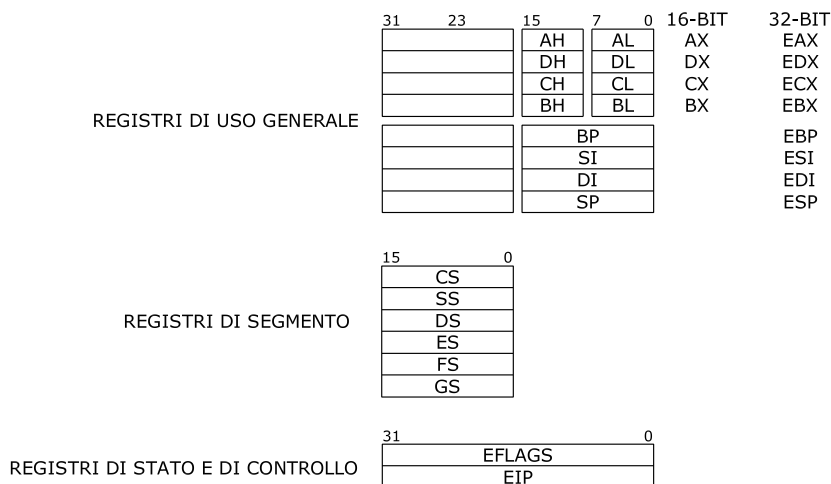


Figura C.7 Il modello di programmazione del 386. I registri hanno la stessa denominazione dei corrispondenti registri dell'8086, ma sono preceduti da una E, a indicare "Esteso a 32 bit". I registri dati a 32 bit sono sostanzialmente equivalenti. I registri di segmento sono 6, sempre a 16 bit. A partire dal 486 sono stati aggiunti i registri per le operazioni in virgola mobile; successivamente le estensioni MMX e SSE.

gistri di segmento che restano a 16 bit. Inoltre, i registri dati sono stati resi (quasi) completamente equivalenti tra loro.

Il modello di Figura C.7 è rimasto comune a tutte le CPU a 32 bit. Processori successivi al 386 hanno introdotto estensioni che hanno potenziato le capacità di elaborazione numerica (integrazione dell'unità aritmetica in virgola mobile e estensione del repertorio con istruzioni SIMD; si veda la Figura C.11).

Indirizzamento dei dati

In Figura C.8 vengono mostrate le possibilità di indirizzamento dei dati. Si osservi che ora i registri dati sono intercambiabili tra loro. Si confronti lo schema di Figura C.8 con quello di Figura C.5.

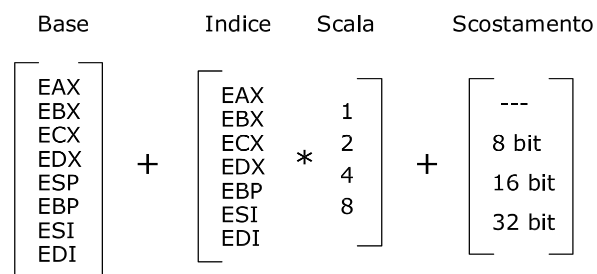


Figura C.8 Indirizzamento dei dati nella architettura a 32 bit. ESP non può essere usato come indice. Quando ESP o EBP vengono usati come base, viene assunto SS come registro di segmento, negli altri casi viene assunto DS (è però possibile imporre un altro registro di segmento attraverso l'uso del prefisso di segmento).

Esempio

Si consideri l'istruzione

```
mov   eax,es:v(ebx)
```

In essa è stato specificato che nel calcolare l'indirizzo deve essere preso `es` come registro di segmento, in luogo del registro `ds` che sarebbe implicito nell'operazione. L'istruzione prevede il registro `ebx` come base; esso sarà codificato nel campo SIB. Infine lo scostamento della variabile `v` (rispetto alla prima posizione nel segmento) sarà codificato nel campo "Scostamento". Non è presente un immediato. Se l'istruzione fosse stata `mov eax,,es:v(ebx)(esi+1278)`, il campo SIB specificherebbe anche `si` e un fattore di scala pari a 4, essendo il riferimento a una parola (4 byte). Infine il campo "Immediato" conterrebbe 1278.

Formato delle istruzioni

La Figura C.9 mostra il formato delle istruzioni dell'architettura a 32 bit. Il campo del codice di operazione può occupare da 1 a 3 byte. Ad esso possono seguire (a seconda dell'operazione) i due byte Mod/RM e SIB, i quali possono anche non essere presenti. Pure i campi di scostamento (indirizzo) e immediato possono essere non presenti o avere dimensioni da 1 a 4 byte. Infine, si noti che possono essere presenti fino a 4 prefissi di diverso tipo; il loro ordine non è rilevante. Essi servono a modificare la normale interpretazione dell'istruzione che segue. Ad esempio, il prefisso di "Cambio di segmento" (*Segment override*) serve a specificare un differente segmento rispetto a quello implicito nell'istruzione.

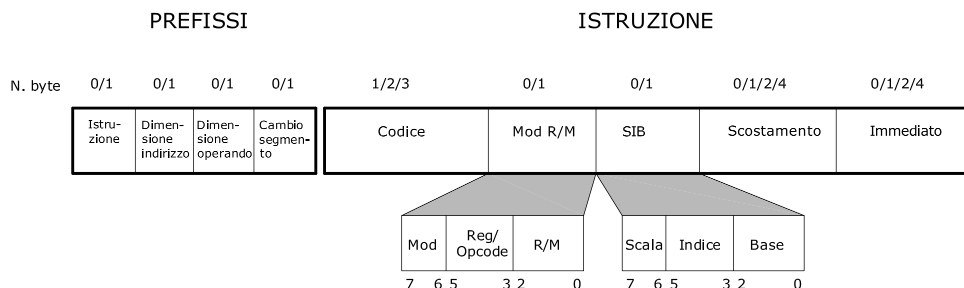


Figura C.9 Formato delle istruzioni dell'architettura a 32 bit. È un classico formato CISC. Il significato dei campi è condizionato dal contenuto di altri campi. Il campo MOD R/M determina quali registri sono implicati; il campo SIB determina il fattore di scala nell'accesso ai dati. Si notino i possibili 4 diversi prefissi. Essi influenzano l'interpretazione di quel che segue. Si veda anche la discussione al Capitolo 3 del testo.

Bus esterni

Il 386 presentava un bus dati e un bus indirizzi a 32 bit. Un bus degli indirizzi a 32 bit consente un indirizzamento fisico massimo di 4 Gbyte (a fronte di 64 TB virtuali). A partire dal Pentium Pro (1995) il bus degli indirizzi è stato esteso a 36 bit in modo da indirizzare (attraverso la cosiddetta PAE, *Physical Address Extension*) fino a 64 GB. Il bus dei dati era stato portato a 64 bit con il Pentium (1993).

Gestione della memoria

Il 386 ha introdotto la paginazione a valle della segmentazione.

Il modello di memoria virtuale dell'architettura a 32 bit è stato descritto al Paragrafo 7.7 del libro. Richiamiamo qui alcuni concetti utili al confronto con la versione a 64 bit. Si ricorda che la memoria virtuale è disponibile solo in modalità protetta, ovvero in quello che viene anche definito modo di funzionamento *nativo*. In Figura C.10 viene mostrato il descrittore di segmento; il significato dei campi è il seguente:

- Il campo "Limite" (20 bit) definisce la dimensione del segmento in base al valore del bit G (granularità).
  - Se G=0 il segmento può avere una dimensione da 1 B a 1 MB, con incrementi di 1 byte.
  - Se G=1 il segmento può avere una dimensione da 4 KB a 4 GB, con incrementi di 4 KByte.

  Al tempo di esecuzione la logica di CPU verifica che l'indirizzamento resti entro i limiti del segmento, in caso contrari viene generata un'eccezione.
- Il campo "Base" (32 bit) fornisce la posizione 0 del segmento nello spazio di 4 GB.
- Il bit S indica se si tratta di un segmento di sistema (S=0) o di un normale segmento di codice o dati[14] (S=1). Il bit S determina l'interpretazione del campo Type.
- Per i segmenti non di sistema il campo Type definisce le modalità di accesso. Nel libro di testo, la Figura 7.18 mostra solo che il bit meno significativo del campo indica se il segmento ha ricevuto almeno un accesso. Per i segmenti di sistema Type ne individua semplicemente il tipo; tra i descrittori di sistema ci sono le tabelle LDT, le porte di interruzione, le porte di chiamata e i segmenti di stato dei task.
- Il campo DPL stabilisce il livello di privilegio del descrittore stesso (*Descriptor Privilege Level*).

[14] Si evidenzia che anche lo stack è da considerare come un segmento dati.

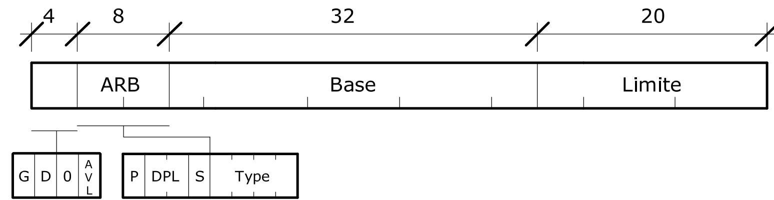


Figura C.10 Formato dei descrittori di segmento. Un descrittore occupa 64 bit (8 byte); i campi Base e Limite che qui sono mostrati in modo unitario, nella pratica sono formati da più sottocampi non contigui (per compatibilità con la precedente architettura a 16 bit).

- Il campo D determina se indirizzi e operandi sono a 32 bit (D=1) o 16 bit (D=0)[15]. D deve essere posto a 1 per segmenti di codice e dati a 32 bit e 0 per segmenti di codice e dati a 16 bit. Attraverso il bit D i processori a 32 bit hanno mantenuto la compatibilità con il precedente mondo a 16 bit[16].

Come detto qui sopra, nel funzionamento protetto un processore può lavorare con indirizzi e operandi a 32 o a 16 bit. Con indirizzi e operandi a 32 bit l'indirizzo lineare può essere al massimo pari a $FFFFFFFF_{16}$ ($2^{32}-1$); gli operandi possono essere da 8, 16, o 32 bit. Con indirizzi e operandi a 16 bit, l'indirizzo lineare (Offset nel segmento) è al massimo $FFFF_{16}$ ($2^{16}-1$); gli operandi possono essere da 8, o 16 bit.

Con l'architettura a 32 bit è stata introdotta la paginazione. La paginazione è a valle della segmentazione. Mentre la segmentazione è inestricabilmente legata alla struttura interna della macchina e quindi ineliminabile, la paginazione può o meno essere abilitata.

C.3.3 Considerazioni conclusive sull'architettura 32 bit

Molti sono stati i modelli di processori con architettura a 32 bit. Ogni nuovo modello ha introdotto miglioramenti tecnologici e architetturali. Come al solito rinviamo ai manuali dei costruttori. A titolo di esempio, in Figura C.11 viene riportato il modello di programmazione del Pentium III; rispetto al modello di programmazione del 386 di Figura C.7, sono stati aggiunti i registri dell'unità in virgola mobile, quelli delle estensioni MMX e quelli relativi alle SSE.

Particolarmente importante è stato il Pentium Pro, con il quale vennero messe in atto alcune innovazioni che hanno finito col marcare tutti gli sviluppi futuri; in particolare il Pentium Pro introduceva la traduzione (interna, al tempo di esecuzione) del repertorio CISC ×86 in un repertorio stile RISC, in modo che il motore di esecuzione potesse operare come una macchina RISC. Del Pentium Pro e del Pentium 4 si parla nel libro di cui questa è un'appendice.

C.3.4 Digressione: il concetto di microarchitettura secondo Intel

Al Capitolo 1 (Introduzione) è stato definito il concetto di architettura. Esso corrisponde alla visione che ha il programmatore assembler, ovvero al repertorio di istruzioni accoppiato al modello di programmazione.

[15] In realtà questa schematizzazione è semplificata; a seconda del tipo di segmento, D tiene anche conto di come esso si espande.

[16] Nella pratica il rispetto di tale compatibilità è risultato più un impiccio che un vantaggio. I sistemi operativi Windows e Linux hanno semplicemente tralasciato tale aspetto e sono buttati subito nel mondo a 32 bit.

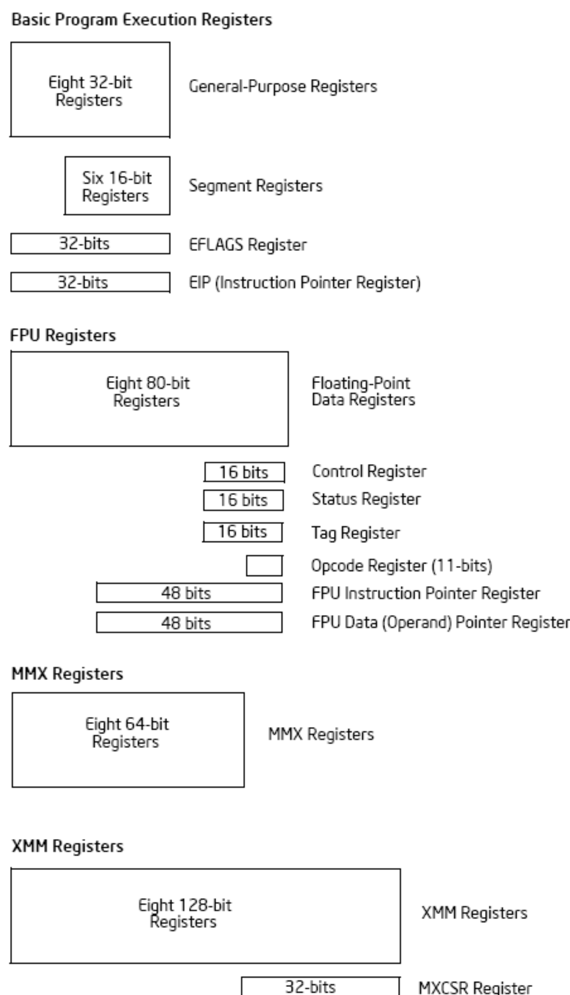


Figura C.11 Modello di programmazione dell'architettura a 32 bit di Intel a partire dal Pentium III. La figura è riportata senza modifiche dai manuali Intel. La FPU, ovvero il precedente coprocessore 8087, è stata integrata nella CPU a partire da 486; le estensioni MMX (*Multi Media eXtensions*) sono state introdotte con il Pentium MMX, le estensioni SSE (*Streaming SIMD Extensions*) sono state introdotte con il Pentium III.I registri MMX sono stati introdotti con il Pentium MMX; i registri XMM sono stati fanno parte delle cosiddette SSE (*Streaming SIMD Extensions*) introdotte con il Pentium III. Ad esse corrisponde un insieme di istruzioni SIMD (*Single Instruction Multiple Data*), ovvero singole istruzioni che operano su più dati contemporaneamente. Qui ci si riferisce ai processori Intel. I processori AMD presentano sostanzialmente le stesse caratteristiche; per esempio AMD chiama "3DNow!" le estensioni corrispondenti alle SSE; le 3DNow! apparvero circa un anno prima delle SSE (queste ultime sono state la risposta di Intel alle 3DNow! di AMD).

In Figura C.12 viene evidenziato il concetto di microarchitettura. La microarchitettura è l'implementazione dell'architettura. La figura mostra quattro microarchitetture corrispondenti all'architettura IA-32 (non vengono mostrati né il 386, né il 486).

Una microarchitettura può dar luogo a differenti prodotti, ovvero a processori che si basano sulla medesima implementazione, ma differiscono tra loro, come, ad esempio il Pentium Pro e le sue evoluzioni Pentium II e Pentium III.

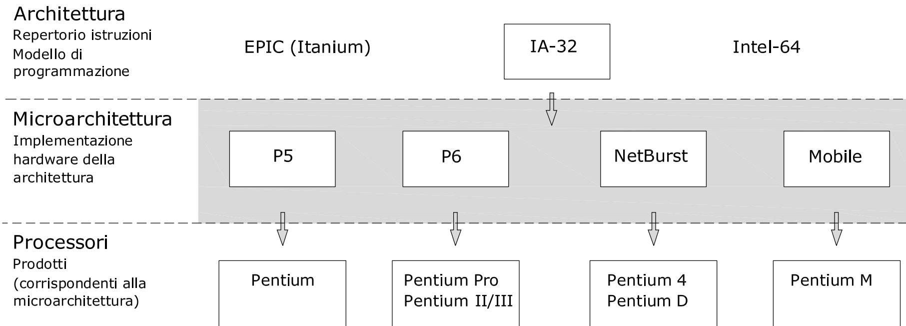


Figura C.12 Il concetto di microarchitettura.

C.4 Architettura a 64 bit

Quando nel 2003 l'AMD introdusse il processore Opteron, estendendo a 64 bit l'architettura ×86, si trattò di una sorta di fulmine a ciel sereno. La nuova architettura venne chiamata AMD64. Fino a quel punto, il gigante Intel proponeva la cosiddetta architettura IA-64, ovvero i processori denominati Itanium, sviluppati in collaborazione con HP, comparsi sul mercato nel 2001. Gli Itanium erano macchine a 64 bit che mantenevano la compatibilità con il mondo ×86 attraverso la simulazione e quindi in modo non efficiente. Al contrario, la caratteristica fondamentale del processore Opteron era quella di consentire l'esecuzione del precedente software a 32 bit in modo "nativo", ovvero senza alcuna penalizzazione, e di consentire l'esecuzione del (nuovo) software a 64 bit pure in modo nativo. In altre parole, l'AMD aveva reso possibile il passaggio a 64 bit, mantenendo la diffusissima architettura ×86, permettendo agli utenti di salvaguardare l'immenso patrimonio software esistente, senza pagare alcun dazio.

L'Intel si dovette adeguare a seguire la strada tracciata da AMD. Del resto i processori Itanium, di cui si parla all'Appendice D, non hanno mostrato di possedere le prestazioni attese. Da circa il 2010 molte società, tra cui Microsoft, hanno abbandonato la linea Itanium. Dal 2007 i manuali di Intel sono intitolati "Intel 64 and IA-32 Architectures"; si tratta sostanzialmente dei precedenti intitolati "IA-32 Intel Architecture", con le dovute estensioni. Chi in precedenza avesse fatto una ricerca sui siti Intel per "64 bit architecture" sarebbe approdato esclusivamente a pagine relative all'architettura Itanium.

In aggiunta alle ovvie modifiche circa l'aumento del parallelismo, la trasformazione a 64 bit dell'architettura ×86 ha essenzialmente riguardato le due seguenti questioni.

- Il mantenimento della compatibilità con il precedente mondo a 32bit.
- Il sostanziale abbandono del modello di memoria segmentata a favore del modello lineare.

C.4.1 Modalità di funzionamento a 64 bit

La compatibilità con l'architettura a 32 bit ha richiesto l'aggiunta di una nuova modalità operativa a quelle di Figura C.6, come illustrato in Figura C.13, dove appare la modalità "IA-32e" ("e" sta per *estesa*). La Figura C.13 è riportata dai manuali Intel. Nel seguito, sebbene si faccia sostanzialmente riferimento ai manuali Intel, adotteremo la dizione di AMD, cioè "modalità Long", anziché "modalità IA-32e" di Intel.

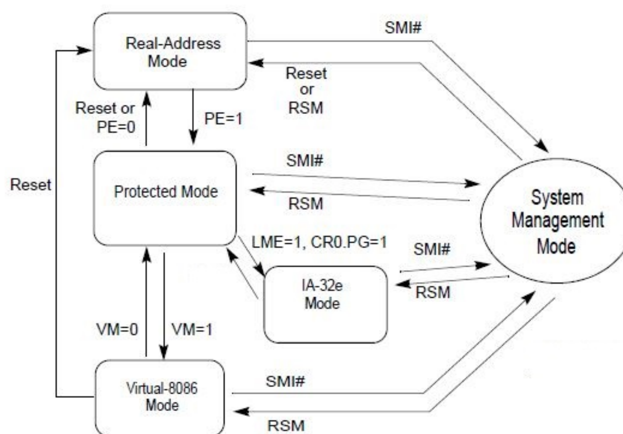


Figura C.13 Stati di funzionamento delle macchine a 64 bit. La figura è riportata senza modifiche dai manuali Intel). Lo stato aggiuntivo viene indicato "modalità IA-32e" da Intel e "modalità Long" da AMD. Nel testo useremo questo secondo termine, perché più evocativo.

Il passaggio alla modalità Long può avvenire solo dal modo Protetto (cioè dal modo "nativo") e viceversa. Si passa dalla modalità protetta alla modalità IA-32e asserendo il bit LME (*Long Mode Enable*); tale bit fa parte di uno speciale registro di controllo denominato EFER, introdotto con l'architettura a 64 bit. La Figura C.13 riporta solo l'atto finale del passaggio al modo Long (asserzione del bit LME del registro di controllo EFER e abilitazione della paginazione); in realtà il passaggio è più complesso, in quanto coinvolge la predisposizione della paginazione a 64 bit (avente un maggior numero di livelli rispetto a quello del funzionamento a 32 bit). Analogamente il passaggio dalla modalità Long alla modalità Protetta richiede altre azioni oltre a riportare a 0 i due bit in questione.

La modalità Long ha due sottomodalità:

a) modalità a 64 bit (*64-bit mode*);
b) modalità compatibile (*compatibility mode*).

La sottomodalità a 64 bit presuppone un sistema operativo a 64 bit, sotto cui eseguono applicazioni a 64 bit che vedono uno spazio lineare ampio 2^{64} bit (vedere più avanti). Il numero dei registri di uso generale passa da 8 a 16 e la loro dimensione è di 64 bit.

La sottomodalità compatibile consente alle precedenti applicazioni a 32 o 16 bit di girare sotto un sistema operativo a 64 bit. Senza che ci sia bisogno di ricompilarle. Le applicazioni vedono solo i primi 4 GB dello spazio lineare. L'indirizzamento può essere a 16 o 32 bit, come esposto per l'architettura $\times 86 - 32$.

Un ulteriore bit del registro EFER, il bit LMA, dice quando il processore è in modalità Long. Quando il processore è in modalità Long, la distinzione tra funzionamento a 64 bit o compatibile è determinata dal bit L del registro CS (si veda più avanti al Paragrafo C.4.3 e in particolare la

Figura C.16. Se L= 0 la macchina si comporta come previsto dall'architettura a 32 bit; se L= 1 come una macchina a 64 bit. Il fatto che la differenza sia basata solo sul bit L del descrittore di codice consente di far girare un sistema operativo a 64 bit che, se del caso, può predisporre segmenti per la modalità compatibile e passare a tale modalità saltando al codice di un segmento compatibile.

Formato delle istruzioni
La compatibilità ha riguardato anche il formato delle istruzioni, rimasto sostanzialmente invariato, ma con la previsione di un nuovo prefisso per trattare i registri a 64 bit. Come si vede dalla Figura C.14 tutti i campi del formato a 32 bit sono rimasti invariati, compreso il campo dello scostamento. Si aggiunge solo il prefisso REX con il quale si denotano i registri a 64 bit. Ovviamente l'assemblatore/il compilatore deve provvedere a inserire automaticamente il prefisso REX quando si indirizzano i registri a 64 bit.

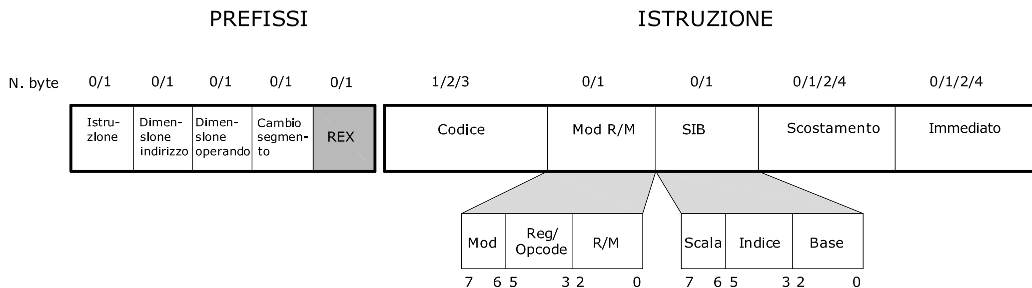


Figura C.14 Formato delle istruzioni funzionamento a 64 bit. Rispetto alla Figura C.9 compare un nuovo prefisso, denominato REX, pure opzionale e che, se presente, deve necessariamente seguire gli altri eventuali prefissi e precedere immediatamente il codice operativo. Notare che i gli altri campi hanno dimensioni invariate. In particolare il campo scostamento è rimasto a 32 bit. Tuttavia con l'indirizzamento indiretto attraverso i registri è visibile tutto lo spazio teorico di 2^{64} KB.

C.4.2 Modello di programmazione

In Figura C.15 viene mostrato il modello di programmazione a 64 bit, aritmetica floating ed estensioni SIMD escluse.

I registri dati (definiti ora registri di uso generale) sono stati portati a 16, aggiungendone 8 a 64 bit ed estendendo i precedenti 8. Il registro IP (denominazione 8086) è stato portato a 64 bit e denominato RIP. Poiché, come vedremo al Paragrafo C.4.3, il modello di memoria a 64 bit è piatto, RIP diventa l'effettivo Program Counter con il quale è possibile indirizzare le istruzioni in tutto lo spazio di 2^{64} KB. Similmente, il puntatore allo stack (RSP) è di 64 bit.

C.4.3 Modello di memoria piatto

La grande differenza tra la versione a 64 bit rispetto alle precedenti sta nell'adottare un modello di memoria lineare, abbandonando il modello segmentato. Quest'ultimo è stato forse l'aspetto più caratteristico dell'architettura ×86, dettato a suo tempo da ragioni pratiche (consentire indirizzi fisici su 20 bit con registri interni a 16) e teoriche (realizzare uno schema di protezione a più livelli, seguendo la via tracciata dal sistema Multix). Nel corso degli anni, il modello segmentato non è stato sfruttato come avrebbe potuto essere: sia Windows sia le varie derivazioni

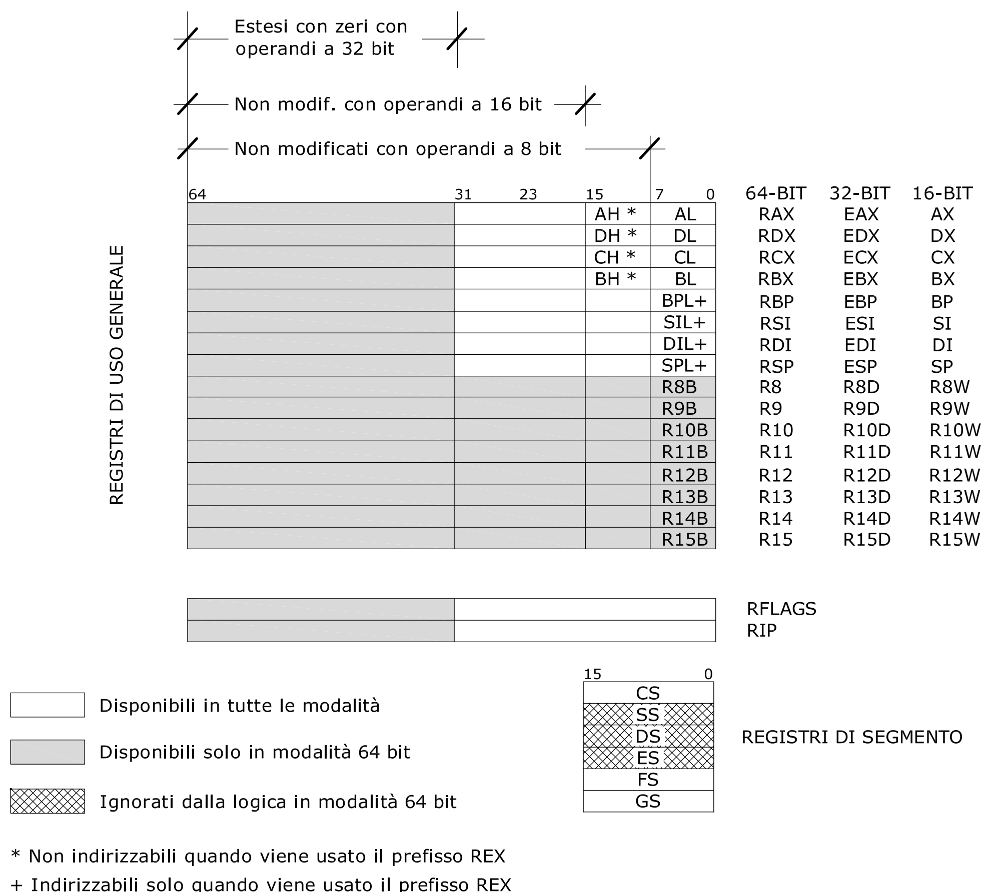


Figura C.15 Il modello di programmazione a 64 bit (non sono rappresentate l'aritmetica floating e le estensioni SIMD). Come si vede esistono alcune limitazioni circa l'uso dei registri: i registri AH, BH, CH e DH non sono indirizzabili in presenza di un prefisso REX, mentre i registri BPL, SIL, DIL e SPL sono indirizzabili solo in presenza di un prefisso REX. Si noti come viene modificato il contenuto dei registri (quali campi sono influenzati) quando gli operandi sono di dimensione inferiore a 64 bit. Per quanto si riferisce ai registri di segmento, in modalità 64 bit, CS punta permanentemente alla posizione di indirizzo 0, mentre DS, ES, SS sono ignorati; FS e GS vengono usati in alcune situazioni per l'indirizzamento. In modalità compatibilità il modello di programmazione resta quello di Figura C.7, ovvero la parte non ombreggiata per i registri di uso generale e tutti i registri di segmento.

di Unix (Linux, Solaris, ..) adottano un modello lineare. Si veda il Paragrafo 7.9 del libro, dove si descrive la memoria virtuale di Linux.

Prima dell'architettura a 64 bit c'era un semplice modo per trasformare un modello segmentato ×86 in un modello lineare piatto: mantenere zero in tutti i registri di segmento. In tal modo si avevano 6 segmenti di 4 GB sovrapposti, ma distinti, e venivano mantenuti tutti i meccanismi di protezione ad anelli. Il sistema operativo Linux, per esempio, fa distinzione tra "spazio utente" e "spazio kernel", il secondo essendo quello del sistema operativo stesso, il primo quello in cui girano i programmi dell'utente. Per ambedue sono previsti un segmento di codice (CS) e uno dati

(DS) (Paragrafo 7.9 del libro).

L'architettura a 64 bit ha apportato una semplificazione ulteriore. Nel funzionamento a 64 bit la logica di macchina tratta i registri di segmento CS, DS, ES, SS come se la base del segmento fosse 0 indipendentemente da ciò che può trovarsi nel campo base del descrittore (si la Figura C.15). Ciò dà luogo a uno spazio di memoria piatto di 2^{64} posizioni per codice, dati e stack[17].

Nel funzionamento in modo compatibile la segmentazione opera esattamente come descritto per l'architettura a 32 bit (Paragrafo C.3). Poiché in modalità Long è possibile sia il funzionamento a 64 bit sia il funzionamento compatibile, è necessario distinguere tra i due; la distinzione avviene tramite il bit L del segmento di codice, come descritto qui di seguito.

Nella modalità a 64 bit i segmenti di codice continuano a esistere, ma sono considerati solo i campi corrispondenti al byte ARB e ai 4 bit di sinistra, come in Figura C.16.

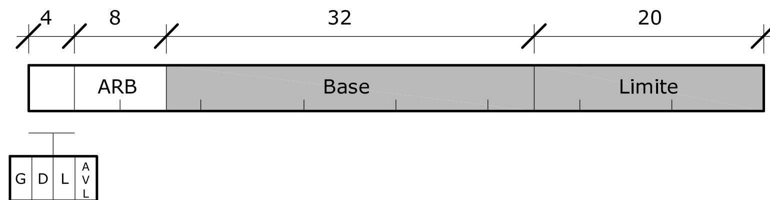


Figura C.16 Descrittore di segmento di codice nell'architettura a 64 bit. La parte in grigio non viene considerata. Il bit L indica se il segmento deve essere inteso per la compatibilità con i 32 bit (L=0) o si tratta di un segmento per i 64 bit (L=1). Nel caso in cui sia L=1, il campo Base viene preso come 0 e il campo Limite non viene considerato. Si evidenzia che nelle precedenti rappresentazioni dei descrittori di segmento (per esempio in Figura C.10), al posto di L è stato sempre riportato 0.

- Quando L= 0 si ha a che fare con un segmento compatibile. In tal caso il bit D indica se se la dimensione standard di dati e indirizzi è 16 (D= 0) o 32 bit (D= 0). Più precisamente, in questa modalità indirizzi e dati sono esattamente come per la tradizionale architettura a 32 bit. Si veda quanto detto per la gestione della memoria al Paragrafo C.3.2.
- Quando L= 1 si ha a che fare con un segmento per il modo 64. In tal caso il bit D può contenere solo 0 e indica che la dimensione standard dei dati è 32 bit, mentre quella degli indirizzi è 64 bit.

Sebbene nel funzionamento a 64 bit la segmentazione appaia come disabilitata, nelle operazioni di caricamento dei registri di segmento la logica di CPU continua a effettuare i test che vengono effettuati nel caso tradizionale, sebbene il loro risultato non abbia rilevanza per il modo a 64 bit. I controlli nel caricamento dei registri di segmento si rendono necessari perché un registro di segmento caricato in modo 64 bit può essere per una applicazione a 32 bit funzionante in modo compatibile.

Si ricorderà che il campo Limite serviva al controllo del superamento della dimensione. nel funzionamento a 64 bit, con lo spazio di memoria piatto, tale controllo non viene più eseguito per nessuno dei 6 registri di segmento[18].

[17] Fanno eccezione FS and GS che possono contenere una base non nulla. FS e GS vengono impiegati per usi speciali dai sistemi operativi. Per esempio Windows a 64 bit impiega GS per puntare a strutture dati (*Thread Environment Block*), nelle quali sono tenute le informazioni di stato dei thread. Linux ne fa un altro uso.

[18] In realtà, almeno AMD ha reintrodotto il controllo del superamento del limite. Intel ha introdotto un meccanismo di verifica dei confini associato alla paginazione.

Nella modalità a 64 bit l'*effective address* è su 64 bit ed esso corrisponde all'indirizzo lineare, essendo la base zero. L'indirizzo viene riportato a 64 bit estendendo il campo dello scostamento ed eventualmente sommandoci il contenuto di un registro di base o indice (a 64 bit). In modo 64 bit sono sempre possibili indirizzi su 32 o 16 bit, ma essi vanno sempre a cadere nei 4 GB inferiori.

C.4.4 La paginazione a 64 bit

Con indirizzi virtuali (lineari) su 64 bit l'albero di traduzione degli indirizzi di infoltisce. In Figura C.17 se ne dà un esempio. La figura assume che l'indirizzo virtuale sia limitato a 48 bit e che le pagine sia di 4 KB. La dimensione dell'indirizzo virtuale come pure il numero di bit del bus indirizzi vengono limitati rispetto a 64 a seconda del modello di CPU.

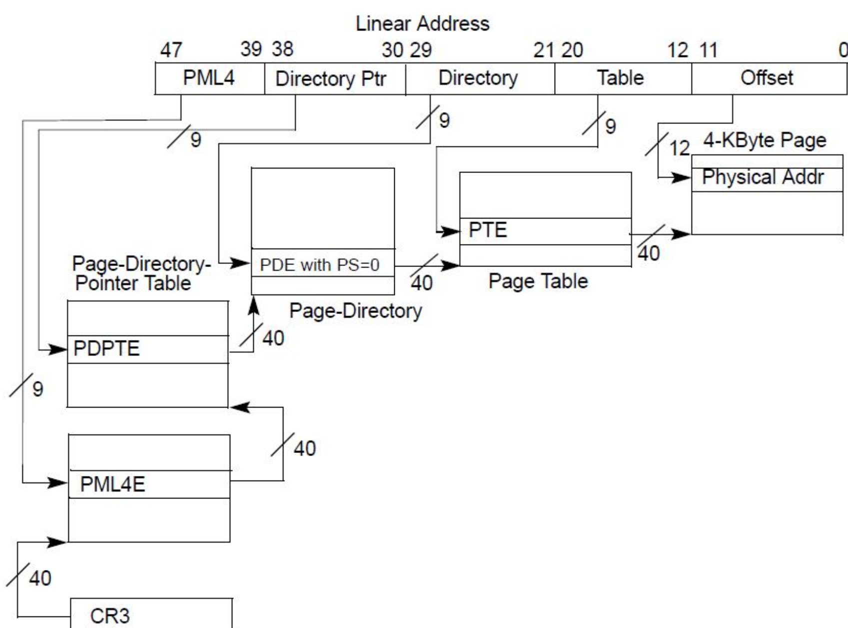


Figura C.17 Esempio di paginazione in modalità Long, per indirizzi virtuali limitati a 48 bit con pagine di 4 KB. Figura tratta direttamente dai manuali Intel.

C.4.5 Considerazioni conclusive sull'architettura 64 bit

L'architettura $\times 86-64$ rappresenta il punto di arrivo di una evoluzione che ha preso le mosse dal processore 8086 nel 1978.

Ci piace sottolineare il fatto che con la modalità a 64 bit è stato abbandonato il tradizionale modello di memoria segmentato. I programmatori in linguaggio Assembler possono tirare un sospiro di sollievo: le complicazioni che la segmentazione comporta sono scomparse!

Alla data di scrittura di queste righe (2017), si è arrivati alla VII generazione di processori Core; il nome di codice di questa generazione è Kabylake. Essa è stata annunciata nella seconda metà del 2016, mentre i primi dispositivi sono usciti agli inizi del 2017. In Figura C.18 si mostra il modello di programmazione corrente comprensivo delle ultime estensioni.

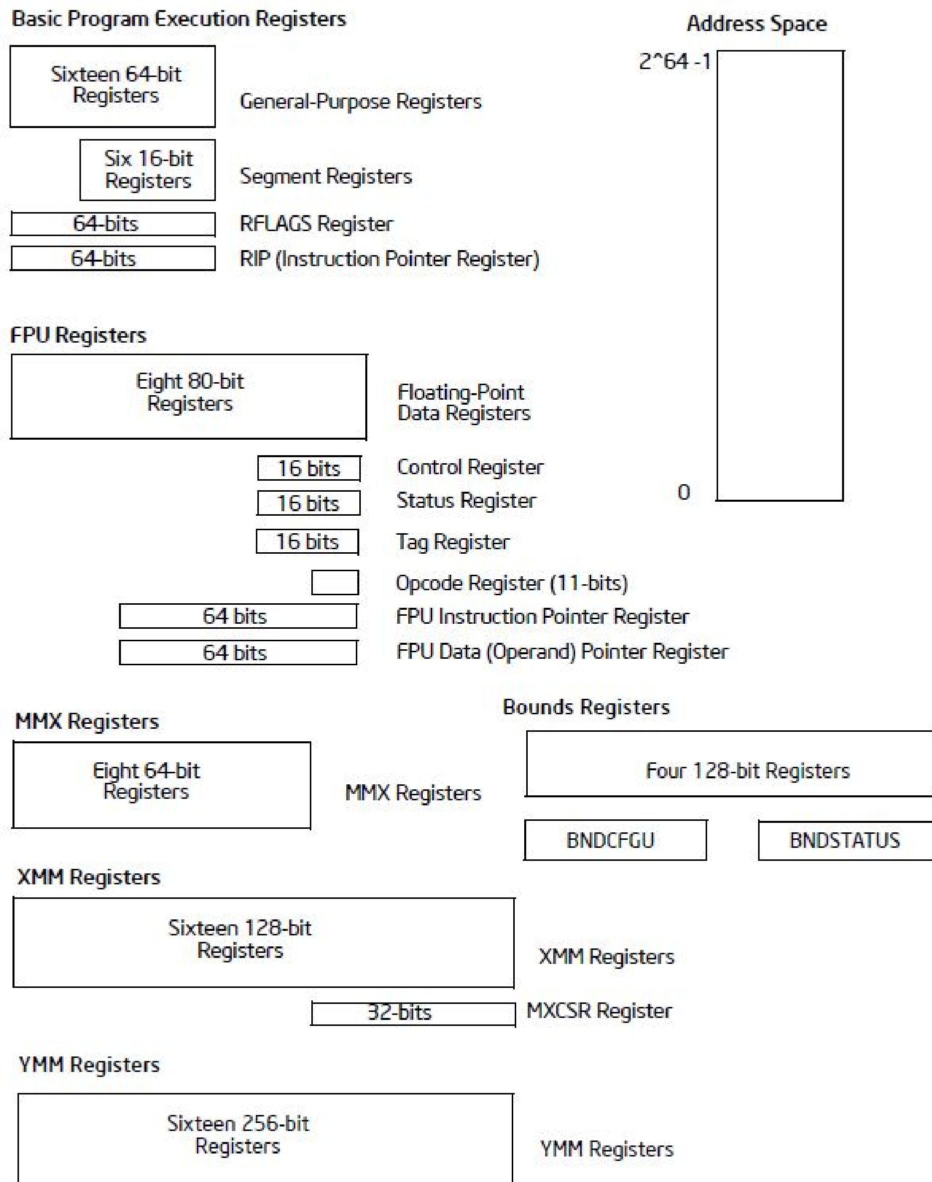


Figura C.18 Modello di programmazione dei processori Intel a 64 bit (tratta dai manuali Intel del 2016). I 4 registri di confine (*Bound register*) sono stati introdotti con il modello Skylake (sesta generazione dei processori Core). Essi servono per quella che Intel definisce MPX (*Memory Protection Extensions*), un miglioramento alla protezione della memoria che permette il riconoscimento di puntatori che vanno oltre i confini entro cui dovrebbero operare. I registri YMM fanno parte delle cosiddette *Advanced Vector Extension*, una ulteriore estensione con operazioni tipo SIMD al repertorio di istruzioni (nel futuro è previsto che questi registri possano diventare di 512 o 1024 bit).

C.4.6 Tick-tock

Con la locuzione tick-tock l'Intel ha designato dalla metà del primo decennio del 2000 fino al 2016 la cadenza di innovazione nello sviluppo dei propri microprocessori e dei dispositivi ad essi collegati. Il tick-tock viene schematizzato in Figura C.19.

Processo tecnologico

45 nm		32 nm		22 nm		14 nm	
Penryn (Core) Nuovo processo	Nehalem Nuova micro architet.	Westmere (Nehalem) Nuovo processo	Sandy Bridge Nuova micro architet.	Ivy Bridge (Sandy Bridge) Nuovo processo	Haswell Nuova micro architet.	Broadwell (Haswell) Nuovo processo	Skylake Nuova micro architet.
Tick	Tock	Tick	Tock	Tick	Tock	Tick	Tock
2007	2008	2010	2011	2012	2013	2014	2015

Figura C.19 Il modello tick-tock dell'evoluzione dei processori Intel. Gli avanzamenti architetturali sono evidenziati in grigio. Gli avanzamenti nel processo riportano tra parentesi l'architettura a cui si riferiscono. Nell'arco di tempo considerato, i tick e i tock si sono susseguiti praticamente uno all'anno. La difficoltà a mantenere un tale ritmo a fatto abbandonare questo modello nel 2016.

Tick e tock designavano le due fasi del processo. Più specificatamente:

- La fase tick aveva lo scopo di migliorare il processo produttivo, ad esempio aumentando la densità dei transistori nei chip, riducendo i consumi di energia, accrescendo il livello delle prestazioni, rispetto all'ultima versione della microarchitettura. Ogni tick mira a realizzare le aspettative che derivano dalla legge di Moore.
- La fase di tock aveva lo scopo di ottenere miglioramenti attraverso avanzamenti architetturali, attraverso l'integrazione di nuove funzionalità resa possibile dal precedente miglioramento del processo produttivo

A partire dal 2016 Intel ha abbandonato questo modello, poiché, a quanto sembra, non più sostenibile. Il nuovo modello di sviluppo denominato PAO prevede ora tre fasi *Process, Architecture, Optimization.*

C.5 Siti web

In questa appendice abbiamo preferito non riportare alcuna bibliografia, preferendo rinviare ai siti dei costruttori `http://www.intel.com`, ovvero `http://www.amd.com`.

Al momento della scrittura di queste righe (fine 2016), Intel esponeva una serie di tre manuali intitolata *Intel 64 and IA-32 Architectures Software Developer's Manual*, per complessivi 5 tomi e un totale di circa 4.600 pagine. Sempre sui siti dei produttori si trovano anche istruttivi *tutorial*, *application note* e simili.

Il lettore è invitato a scaricare dal sito dei produttori i manuali menzionati e confrontare quanto è stato riportato con il loro contenuto.

La lista di tutti i microprocessori prodotti da Intel, a partire dal micro a 4 bit 4004 (inizi anni '70) fino all'ultimo annunciato, si trova alla pagina:
`http://www.intel.com/pressroom/kits/processors/quickref.htm`.

Coloro che fossero interessati ai manuali di processori o di dispositivi del passato possono fare riferimento ai siti riportati nella premessa del libro.

Domande ed esercizi

C.1 Si cerchi sui siti Intel o AMD qual è l'ultima CPU introdotta da uno di questi costruttori e si confrontino le relative prestazioni con quelle della CPU contenuta nel PC di proprietà del lettore.

C.2 Si dia una ragionevole spiegazione del fatto per cui nel primo PC venne impiegato un 8088, anziché un 8086, sebbene quest'ultimo fosse in circolazione da prima e avesse prestazioni superiori.

C.3 Il lettore cerchi sui siti Intel o AMD il manuale dell'architettura ×86 e stabilisca quali tra queste istruzioni hanno effetto sulla parola di stato (EFLAGS):

```
ADD    MOV    JMP    JE    PUSH    IMUL.
```

C.4 Si discuta la funzione del bit TF del registro FLAGS e la sua utilizzazione in un ipotetico *debugger*.

C.5 Quali sono i possibili registri di segmento alternativi a CS nei riferimenti corrispondenti a fetch delle istruzioni ?

C.6 Con riferimento all'8086, si discuta l'impiego del registro BP. Quale registro di segmento viene normalmente usato con BP? Quali sono i possibili registri di segmento alternativi? Si prospetti una possibile situazione in cui è conveniente impiegare BP per indirizzare entro lo stack e per indirizzare in un diverso segmento.

C.7 Le operazioni PUSH e POP dell'8086 muovono sempre 16 bit. Se lo stack non è allineato agli indirizzi pari ogni accesso allo stack richiede due cicli di lettura o scrittura anziché uno. Si ipotizzi un programma nel quale ci sia un 20% di istruzioni operanti sullo stack. Di quanto si degradano le prestazioni nel caso di stack non allineato?

C.8 Si illustri l'effetto delle seguenti istruzioni (8086)

```
MOV    AX,127
MOV    AX,127[BX]
MOV    AX,127[BX][SI]
```

C.9 Con riferimento all'8086, si assuma che il registro CS contenga A05B e che il registro IP contenga F00C. A quale indirizzo fisico si trova la prossima istruzione eseguita?

C.10 Con riferimento all'8086, si assuma che DS contenga 104A, che BX contenga 200A e che SI contenga 78(H). A quale indirizzo fisico si trova il byte letto dall'istruzione seguente?

```
MOV     AL,[BX][SI]
```

C.11 Si considerino le seguenti istruzioni 8086

```
mov     al,[bx]
mov     al,[bx+9]
mov     al,[si]
mov     al,[si+1024]
mov     al,[bx][si]
mov     al,[bx+9][si]
mov     al,9[bx][si]
mov     al,[bx][si+1024]
mov     al,1024[bx][si]
mov     al,[bx+1033][si]
mov     al,[bx][si+1033]
mov     al,[bx+9][si+1024]
```

Servendosi di un assemblatore (Appendice **??**), si esamini il formato del codice generato per le singole istruzioni. Con riferimento alla Figura 3.3 del libro si richiede cosa viene generato per ciascuna istruzione per i byte del campo `OP CODE` e della modalità di indirizzamento e per gli eventuali altri byte.

C.12 Con riferimento all'8086/88 si assuma che una lettura/scrittura in memoria richieda 4 cicli di clock e si considerino le due istruzioni `PUSH AX` e `POP MEM`.

Si assuma che la dimensione delle due istruzione sia di 16 bit e che le due istruzioni siano allineate (agli indirizzi pari), come pure lo stack. Si faccia l'ipotesi che oltre alle dovute letture/scritture in memoria l'esecuzione delle due istruzioni richieda 3 cicli addizionali per la `PUSH` e 4 cicli addizionali per la `POP`.

Si valuti il numero di cicli di clock richiesto per l'esecuzione delle due istruzioni prima per l'8086 e successivamente per l'8088.

C.13 Si discuta la funzione del registro BP nelle chiamate ai sottoprogrammi, nel passaggio dei parametri e per l'allocazione delle variabili locali. In particolare, si faccia riferimento alla seguente funzione C, assumendo che un intero occupi 16 bit e che la chiamata sia di tipo near:

```
int f(int m, int *n)
{
    int x;      /* variabile locale intera*/
    *n= *n/2;
    x= m + *n;
    return x;
}
```

Si discuta quanto vale BP durante l'esecuzione del corpo della funzione (si assuma che prima della chiamata BP valga BP0). Si schematizzi lo stato dello stack durante l'esecuzione. Si traducano gli statement della funzione in equivalente assembler.

Soluzioni appendice C

Aggiornato il 31 marzo 2017

C.3 Le istruzioni `MOV` , `JMP`, `JE` e `PUSH` non hanno effetto sui bit di stato. Per quanto riguarda le rimanenti, esse hanno l'effetto illustrato qui di seguito.

- L'istruzione `ADD` ha effetto sui flag: OF, SF, ZF, AF, CF,e PF, in accordo con il risultato della somma. Per il loro significato si veda a pagina 7.
- Per quanto riguarda l'istruzione `IMUL` faremo riferimento all'architettura IA-32.

L'istruzione `IMUL`:

- Può operare su registri o locazioni di memoria a 8,16 o 32 bit.
- Ha 3 modalità di impiego, con 1,2 o 3 parametri[1].

Schematicamente le possibili forme dell'istruzione sono:

```
IMUL    SRC
IMUL    DEST,SRC
IMUL    DEST,SRC1,SRC2
```

Nel primo caso `SRC` rappresenta il secondo operando, mentre l'accumulatore `AX` rappresenta il primo operando della moltiplicazione e la destinazione in cui viene scritto il risultato. Nel caso di 2 o 3 operandi, `DEST` è sempre rappresentato da un registro, mentre `SRC` può essere indifferentemente una locazione di memoria o un registro. Alcuni esempi:

```
IMUL    BL              ;AX  <- AL * BL
IMUL    CX,V1           ;CX  <- CX * V1
IMUL    EBX,V2,V3       ;EBX  <- V2 * V3
```

dove V1,V2,V3 sono locazioni di memoria (16 o 32 bit).

Di seguito viene presentato l'algoritmo di generazione dei bit di stato, come rilasciato da Intel.

```
IF (NumberOfOperands = 1)
THEN IF (OperandSize = 8)
    THEN
        AX <- AL * SRC (* moltiplicazione con segno *)
        IF AL = AX
        THEN CF <- 0; OF <- 0;
        ELSE CF <- 1; OF <- 1;
        FI;
ELSE IF OperandSize = 16
     THEN
        DX:AX <- AX * SRC (* moltiplicazione con segno *)
        IF sign_extend_to_32 (AX) = DX:AX
        THEN CF <- 0; OF <- 0;
        ELSE CF <- 1; OF <- 1;
```

[1] Ci stiamo riferendo all'architettura a 32 bit. Nel caso dell'8086 l'istruzione `IMUL` prevede sempre e soltanto un operando che viene moltiplicato con l'accumulatore, dove poi viene scritto il risultato

```
        FI;
ELSE (* OperandSize = 32 *)
    EDX:EAX <- EAX * SRC (* moltiplicazione con segno *)
    IF EAX = EDX:EAX
    THEN CF <- 0; OF <- 0;
    ELSE CF <- 1; OF <- 1;
    FI;
FI;
ELSE IF (NumberOfOperands = 2)
    THEN
        temp <- DEST * SRC (* moltiplicazione con segno;
                             temp dimensione doppia di DEST *)
        DEST <- DEST * SRC (* moltiplicazione con segno *)
        IF temp = DEST
        THEN CF <- 1; OF <- 1;
        ELSE CF <- 0; OF <- 0;
        FI;
    ELSE (* NumberOfOperands = 3 *)
        DEST <- SRC1 * SRC2 (* moltiplicazione con segno *)
        temp <- SRC1 * SRC2 (* moltiplicazione con segno;
                              temp dimensione doppia di SCR1 *)
        IF temp != DEST
        THEN CF <- 1; OF <- 1;
        ELSE CF <- 0; OF <- 0;
        FI;
    FI;
FI;
```

C.4 Il registro FLAGS contiene informazioni di stato e di controllo. Quando vale 1 il bit TF determina un'interruzione (trap) dopo l'esecuzione di ogni singola istruzione. Un ipotetico debugger può utilizzare questo bit al fine dell'esecuzione *single step*. Più specificatamente, per eseguire l'istruzione all'indirizzo I in *single step mode*, il debugger asserisce TF e quindi effettua il salto all'indirizzo I. L'interruzione che segue all'esecuzione dell'istruzione viene raccolta dal debugger stesso, che esaminare lo stato della macchina e mostrare gli effetti determinati dall'esecuzione dell'istruzione all'indirizzo I.

C.5 Nella fase di opcode fetch non può essere usato alcun registro in alternativa al registro CS.

C.6 Il registro BP (Base Pointer) viene usato normalmente come puntatore entro lo stack. Ad esempio:

```
MOV     [BP], BX     ; M[SS:BP] <- BX
```

ovvero il normale uso di BP assume il registro SS come registro di segmento.

BP può anche essere usato relativamente a tutti gli altri registri di segmento (in questi casi viene generato l'adeguato prefisso di segmento). Ad esempio:

```
MOV     DS:[BP], BX     ; M[DS:BP] <- BX
```

Per quanto si riferisce all'ultima parte dell'esercizio, la situazione può essere quella di dover trasferire un blocco di dati da stack a memoria, o viceversa. L'istruzione seguente, inserita dentro un ciclo (che fa incrementare BP) produce tale effetto.

```
MOV     DS:[BP], [BP]     ; M[DS:BP] <- M[SS:BP]
```

C.7 Le prestazioni *rivedere, capire il 20% del testo* sono date dall'inverso del tempo medio richiesto a eseguire le istruzioni, ovvero dall'inverso di $T_{Fetch}+T_{Exe}$. Nel caso stack non allineato c'è un incremento del 20% del tempo della fase di esecuzione, dunque il tempo medio di esecuzione di un'istruzione è dato da $T_{Fetch}+T_{Exe}(1+0,2)$. Assumendo $T_{Fetch}=T_{Exe}=T$, il rapporto delle prestazioni dei due casi è dato da $\frac{2T}{2{,}2T}=0,91$. In conclusione il fatto che lo stack non sia allineato comporta, con questi dati, un degrado delle prestazioni del 9%.

C.8

- `MOV AX,127                    ;AX<-127`
  ovvero l'istruzione mette in AX il valore 127;
- `MOV AX,127[BX]               ;AX<-M[DS: BX+127]`
  ovvero l' istruzione pone in AX il contenuto della memoria all'indirizzo 127 + BX (indirizzamento relativo al registro e scalato);
- `MOV AX,127[SI][BX]           ;AX<-M[DS: 127+BX+SI]`
  ovvero l' istruzione mette in AX il valore di memoria all'*effective address* dato da 127 + BX + SI (indirizzamento relativo al registro BX, con scostamento e indiciato).

C.9 `CS` contiene la base del segmento di codice corrente, mentre `IP` (Istruction Pointer), contiene l'offset dell'istruzione successiva da eseguire. L'indirizzo logico si denota allora come `A05B:F00C`. Il corrispondente indirizzo fisico si ottiene sommando il contenuto di IP con il contenuto di CS moltiplicato per 16. Tenendo conto che moltiplicare per 16 un numero in esadecimale, equivale ad aggiungere uno zero a destra, l'indirizzo fisico si calcola come: `A05B0 + 0F00C = AF5BC`.

C.10 L'indirizzo logico a cui fa riferimento l'istruzione `MOV AL, [BX][SI]` è `104A:200A+78`.
Il corrispondente indirizzo fisico è quindi: `104A0+0200A+00078=12522`.

C.11 Illustreremo solo 3 casi:

- `MOV AL,[BX] ; AL <- M[BX]`
  Il codice prodotto dall'assemblatore è: `8A 07`. In accordo con il manuale INTEL, `8A` è il codice di OPCODE per il `MOV` la cui destinazione è un registro a 8 bit, a partire da una locazione di memoria o un registro a 8 bit. `07` è il codice che identifica la destinazione in `AL` e la sorgente nel valore contenuto nella cella di memoria il cui indirizzo è in `BX`.
- `MOV AL,[BX][SI] ; AL <- M[BX+SI]`
  Il codice prodotto dall'assemblatore è: `8A 00`, dove `8A` ha il significato precedentemente esposto, mentre l'estensione `00` rappresenta `AL` come destinazione e `M[BX+SI]` come sorgente.
- `MOV AL,1024[BX][SI] ; AL <- M[1024+BX+SI]`
  L'assemblatore produce: `8A 80 0400`. Il valore `80` rappresenta `AL` come destinazione e la modalità di indirizzamento; il cui valore `0400` (esadecimale) sta per 1024 decimale.

C.12 Consideriamo prima il caso 8086.

In base alle assunzioni del testo, l'istruzione `PUSH AX` richiede i seguenti passi:

- fetch dell'istruzione (4 cicli)
- esecuzione (3 cicli)
- scrittura del dato (nello stack)(4 cicli)

Con le ipotesi fatte, `PUSH AX` su 8086 viene eseguita in 11 cicli di clock. Per `POP MEM` invece i passi sono:

- fetch dell'istruzione (4 cicli)
- lettura dello stack (4 cicli)
- cicli addizionali di esecuzione (4 cicli)
- scrittura (in memoria) (4 cicli)

Con le ipotesi fatte, l'istruzione `POP MEM` su 8086 richiede quindi 16 cicli di clock.

Consideriamo ora il caso 8088.

In base alle assunzioni del testo, l'istruzione `PUSH AX` richiede i seguenti passi:

- fetch dell'istruzione (8 cicli)
- esecuzione (3 cicli)
- scrittura del dato (nello stack)(8 cicli)

Con le ipotesi fatte, `PUSH AX` su 8088 viene eseguita in 19 cicli di clock. Per `POP MEM` invece i passi sono:

- fetch dell'istruzione (8 cicli)
- lettura dello stack (8 cicli)
- cicli addizionali di esecuzione (4 cicli)
- scrittura (in memoria) (8 cicli)

Con le ipotesi fatte, l'istruzione `POP MEM` su 8088 richiede quindi 28 cicli di clock.

C.13

Si assume che la funzione nel testo, abbia una chiamata dal codice `C` del tipo `k=f(m,*n)`. Denotiamo con BP0 e SP0 il contenuto dei registri BP e SP prima della chiamata della funzione. Seguendo la convenzione C, per la quale vengono caricati prima gli ultimi parametri, lo statement viene tradotto come segue. Si noti che si assume di usare AX per restituire al chiamante il valore calcolato dalla funzione.

```
LEA     AX,n         ;carica l'indirizzo effettivo di n
PUSH    AX           ;deposita il valore nello stack
MOV     AX,m         ;carica in AX il valore a
PUSH    AX           ;deposita il valore nello stack
CALL    f            ;chiama la SubRoutine f
ADD     SP,4         ;riporta SP a prima dei due push precedenti la chiamata
MOV     k,AX         ;assegna il risultato
```

All'ingresso in `f` il contenuto dello stack, per la parte di interesse, è quello mostrato in Figura C.1:

Per indicizzare gli elementi nello stack viene utilizzato BP. Per tale motivo esso deve contenere l'indirizzo di una posizione nota di riferimento che, all'ingresso della routine, è la testa dello stack. L'aggiornamento di BP richiede il preventivo salvataggio del suo contenuto, quindi all'ingresso in `f` vengono eseguite le seguenti istruzioni:

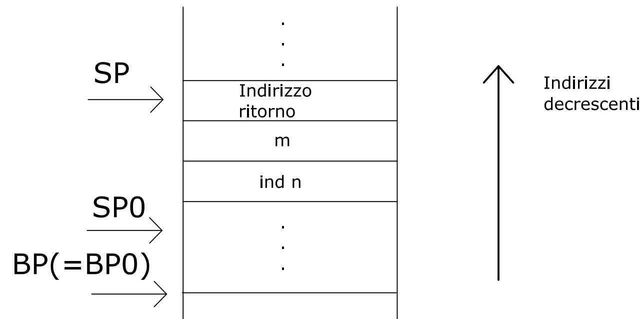


Figura C.1 Stato dello stack immediatamente dopo l'esecuzione dell'istruzione `CALL f`. (Esercizio 14.15).

```
PUSH    BP
MOV     BP,SP
SUB     SP,2        ;spazio per la variabile intera  x
```

Nell'architettura Intel lo stack cresce verso gli indirizzi più bassi, per questo lo spazio per `x` si ottiene con l'istruzione `SUB`. La situazione risultante nello stack è mostrata in Figura C.2.

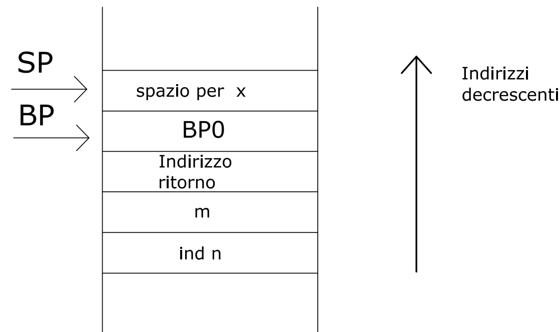


Figura C.2 Stato dello stack dopo l'aggiornamento di BP e l'allocazione dello spazio per il parametro interno x.

Si noti inoltre che all'interno della funzione, i parametri `m,n` e la variabile interna `x` hanno questi scostamenti rispetto al contenuto di BP:

```
    m:  +4
ind n:  +6
    x:  -2
```

Lo statement `*n=*n/2` viene così tradotto:

```
MOV     BX,[BP+6]       ;carica *n
MOV     AX,[BX]         ;AX<-n
DIV     2               ;esegue la divisione per 2
MOV     [BX],AX         ;salva il risultato
```

Il successivo statement `x= m + *n` diventa:

```
MOV     AX,[BP+4]       ;AX<-m
MOV     BX,[BP+6]       ;BX<-*n
ADD     AX,[BX]         ;AX<- m+n
MOV     [BP-2],AX       ;salva in x il risultato
```

Le istruzioni corrispondenti al ritorno dalla funzione (`return x`) sono:

```
MOV     AX,[BP-2]       ;carica in AX il valore x
MOV     SP,BP           ;riporta lo stack
POP     BP              ; come all'atto
RET                     ;  della chiamata
```

Indice analitico

Indice